Хмельницький національний університет
Факультет інформаційних технологій
Кафедра телекомунікацій, медійних та інтелектуальних технологій

КВАЛІФІКАЦІЙНИЙ ПРОЕКТ

Бакалавр
Освітній рівень

Система моніторингу електросамокатів
Назва теми

КПТР.220910.01.12 ПЗ

Галузь знань 17 «Електроніка та телекомунікації»
Шифр і назва
Спеціальність 172 «Телекомунікації та радіотехніка»
Шифр і назва
Освітня програма «Телекомунікації, медійні технології та інтелектуальні мережі»
Назва

Виконав:
здобувач 3 курсу, група ТР2с-22-1 _____ Іван ШВЕЦЬ
підпис Ініціали, прізвище

Керівник: к. техн. наук, доцент _____ Віктор МІШАН
підпис Ініціали, прізвище

До захисту допускаю:

Зав. кафедри: д-р техн. наук, професор _____ Сергій ПІДЧЕНКО
підпис Ініціали, прізвище

«16» 06 2025р.

Хмельницький,2025

Хмельницький національний університет

Факультет інформаційних технологій
Кафедра телекомунікацій, медійних та інтелектуальних технологій
Освітній рівень: бакалавр
Галузь знань: 17 Електроніка та телекомунікації
Спеціальність: 172 Телекомунікації та радіотехніка
Освітня програма: Телекомунікації, медійні технології та інтелектуальні мережі

ЗАТВЕРДЖУЮ
Завідувач кафедрою ТМІТ
Сергій ПІДЧЕНКО
«10» 02 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНИЙ ПРОЕКТ

ШВЕЦЬ Іван Андрійович
Прізвище, ім'я по батькові здобувача

1 Тема роботи: Система моніторингу електросамокатів
Керівник роботи: МІШАН Віктор Володимирович, к. техн. наук, доцент
Затверджено наказом по університету від «07» лютого 2025р. № 20 2 Строк подання здобувачем роботи на кафедру 02 червня 2025 року
3 Живлення контролера: вхідна напруга 48 В, перетворена до 5 В або 3.3 В для живлення мікроконтролера ESP8266, напруга живлення електросамоката: 48 В постійного струму, передача даних: Wi-Fi зв'язок через ESP8266 до серверної частини, гальванічна розв'язка між живленням електродвигуна та обчислювальним модулем, цифрове керування та обробка даних через мікроконтролер з використанням програмного забезпечення
4 Аналіз сучасних систем моніторингу IoT у сфері мікромобільності, аналіз існуючих IoT-платформ для електротранспорту, обґрунтування вибору апаратної архітектури електросамоката (ESP8266, GPS, сенсори), проєктування схеми збору та обробки телеметричних даних, розробка структурної схеми взаємодії пристрою з сервером, алгоритми передачі та обробки даних у хмарному сервісі, розробка інтерфейсу адміністративної панелі з мапою та фільтрами, безпека та захист персональних та системних даних
5 Перелік графічного матеріалу (із зазначенням обов'язкових креслень):
1 Структурна схема IoT-системи моніторингу електросамоката. 2 Принципова схема електроживлення мікроконтролера та периферійних пристроїв. 3 Принцип

роботи модуля зв'язку та обробки даних (ESP8266, GPS, сенсори). 4 Блок-схема роботи програмної логіки з передачею даних на сервер.
6 Консультанти розділів кваліфікаційного проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	СТЕЦЮК Вітор к.т.н., доцент		
Антиплагіат	ПИВОВАР Олег к.т.н., доцент		

7 Дата видачі завдання «10» 02 2025 року

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування виду роботи	Форма звітності, термін виконання	Примітка
1.	Розробка завдання на кваліфікаційний проект	01.02.2025р.	виконано
2.	Складання індивідуального плану на кваліфікаційний проект	15.02.2025р.	виконано
3.	Написання першого (теоретичного) розділу.	10.03.2025р.	виконано
4.	Написання другого розділу	26.03.2025р.	виконано
5.	Написання третього розділу	15.04.2025р.	виконано
6.	Написання четвертого розділу	30.04.2025р.	виконано
7.	Написання вступу і загальних висновків та пропозицій до кваліфікаційного проекту	10.05.2025р.	виконано
8.	Оформлення кваліфікаційного проекту	15.05.2025р.	виконано
9.	Рецензування кваліфікаційного проекту	20.05.2025р.	виконано
10.	Презентаційні матеріали за результатами виконання кваліфікаційного проекту	28.05.2025р.	виконано

Здобувач _____ (підпис, дата) Іван ШВЕЦЬ

Науковий керівник _____ (підпис, дата) Віктор МІШАН

АНОТАЦІЯ

Тема кваліфікаційного проекту: «Система моніторингу електросамокатів»

Автор проекту: ШВЕЦЬ Іван Андрійович

Керівник проекту: МІШАН Віктор Володимирович, к.техн. наук, доцент.

Пояснювальна записка: 90 сторінок, 12 рисунків, 2 таблиці, 29 джерел.

Графічна частина: технічні креслення.

КЛЮЧОВІ СЛОВА: ЕЛЕКТРОСАМОКАТ, IoT, GPS-МОНІТОРИНГ, ТЕЛЕМЕТРІЯ, СИСТЕМА КЕРУВАННЯ, GOOGLE MAPS API, КЛІЄНТСЬКО-СЕРВЕРНА АРХІТЕКТУРА, ADOINISJS, WEBSOCKET.

Об'єктом дослідження є автоматизована система віддаленого моніторингу електросамокатів, що забезпечує відстеження їхнього стану, розташування та параметрів у режимі реального часу.

Метою кваліфікаційного проєкту є розробка функціональної та масштабованої інформаційної системи для моніторингу електросамокатів, яка дозволяє забезпечити ефективне управління парком пристроїв, контроль зон пересування, облік маршрутів, швидкості, рівня заряду батареї та інших технічних параметрів.

Кваліфікаційний проєкт присвячений дослідженню IoT-рішень для мобільного транспорту, розробці клієнт-серверного вебдодатку на основі стеку технологій (AdonisJS, WebSocket, Tailwind, Google Maps API), створенню бази даних, побудові інтерфейсу адміністратора, реалізації телеметрії в реальному часі, а також візуалізації пристроїв на карті з фільтрами та індикаторами стану.

«09» червня 2025р.

№ ряд ка	Форм ат	Позначення	Найменування	К і л ь к .	№ е к з.	Примітка
			Документація загальна			
1	А4		Завдання на дипломний проект	1	1	
2	А4		Анотація	1	1	Укр.
3	А4	КПТР.220910.01.12 ПЗ	Система моніторингу електросамокатів			
			Пояснювальна записка	1	1	84 арк.
4	А4	КПТР.220910.01.12 Е3	Система моніторингу електросамокаті			
			Схема електрична принципова	1	1	1 арк.
5	А4	КПТР.220910.01.12 Д1	Система моніторингу електросамокаті			
			Діаграма UseCase	1	1	1 арк.
6	А4	КПТР.220910.01.12 Д2	Система моніторингу електросамокаті			
			Діаграма UseCase	1	1	1 арк.
7	А4	КПТР.220910.01.12 Д3	Система моніторингу електросамокаті			
			Діаграма Activity	1	1	1 арк.
8	А4	КПТР.220910.01.12 Д4	Система моніторингу електросамокаті			
			Діаграма Communication	1	1	1 арк.

Зм.	Арк.	№ докум.	Підп.	Дата	КПТР.220910.01.12 ВП			
Розроб.		Швець І. А.		13.06	Система моніторингу електросамокатів Відомість дипломного проекту	*Літ.*	*Аркуш*	*Аркушів*
Перевір.		Мішан В. В.		13.06.25		н	1	1
						ХНУ, гр. ТР2с-22-1		
Н .контр.		Стецюк В. І.		16.06.25				
Затв.		Підченко С. К.		16.06.25				

ЗМІСТ

Змін	Аркуш	№ докум.	Підпис	Дата	КПТР.220910.01.12 ПЗ		
Розробив		Швець І.А			Система моніторингу електросамокатів	Літ	Аркуш	Аркушів
Перевірив		Мішан В.В		16.06.25				
Н.контр.		Стецюк В.І		16.06.25		ХНУ, гр. ТР2с-22-1		
Затвер.		Підчинко С.К.		16.06.25				

ВСТУП

Шалений розвиток нових інформаційних технологій протягом XXI століття залишив певний відбиток у становленні особистості сучасної людини. Сильний потік інформації, застосування комп'ютерних технологій у телебаченні, рекламі та і майже у всіх сферах людського життя спричинили великий вплив на виховання людини та її сприйняття навколишнього оточуючого світу.

В ногу із соціальними сайтами активно розвиваються месенджери – системи обміну миттєвими повідомленнями, аудіо-, відео-контентом, документами різних типів. На відміну від сайтів з тією чи іншою інформацією, месенджери не викликають розповсюдженню надлишкової інформації яка знаходиться на інтернет сторінці, адже люди діляться між собою інформацією яка їх цікавить ,а не так як на сторінках на яких розміщена, як і потрібна так і не зовсім потрібна.

Сайти повині приносити користь користувачу при пошуку, але сьогодні ми спостерігаємо зовсім іншу картину: сайти наповнені непотрібною рекламою та непристойним матеріалом. Та і самі власники сайту підтримують їх, перш за все, з інформаційною метою (адже усім відомо, що рекламодавці платять величезні гроші, щоб саме їх реклама „запускалась" на тому чи іншому сайті), а вже потім для вивчення інформації людиною. Ще один недолік соціальних мереж – це створення ефекту „затягування". Але в той же час, не слід бути такими категоричними. Реклама на сайтах певною мірою допомагають самореалізовуватись, розкривати таланти людям, яким не вистачає ресурсів або сміливості проявити себе в реальному житті.

Отже, вкрай важливо вміти відокремлювати потрібну інформацію від надокучливої, яка впливає на пошук в інтернет-мережі і використовувати їх для розвитку своєї персони і своїх знайомих.

Розділ 1 Загальний розділ веб-додатку “Система моніторингу електросамокатів”

1.1 Опис предметної області

Веб-сайт – це будь-яка комп'ютерна програма, яка виконує певну функцію, використовуючи веб-браузер у якості свого клієнта. Додаток може бути таким же простим, як дошка оголошень або контактна форма на веб-сайті, або настільки ж складний, як текстовий процесор або багатокористувацький мобільний ігровий додаток, який ви завантажуєте на телефон.

Веб-додаток звільняє розробника від відповідальності за створення клієнта для певного типу комп’ютера або конкретної операційної системи, тому кожен може використовувати програму разом із наявним доступом до Інтернету. Оскільки клієнт працює у веб-браузері, користувач може використовувати IBM-сумісний або Mac. Вони можуть мати ОС Windows. Вони навіть можуть використовувати Googl Chrome або Opera, хоча для деяких програм потрібен певний веб-браузер.

Проаналізувавши всі можливі варіанти створення веб-додатку для взаємодії з браузером, що для написання такого веб-додатку необхідно використовувати одну з мов веб-програмування: Node.JS, Python, Java. Необхідно визначити яка саме мова найкраще підходить для написання такого веб-додатку. Також є важливим вміння працювати API (Application Programming Interface), які надають доступ до сервера.

І на останок, потрібно визначити мету веб-додатку, оскільки в іншому випадку він не матиме сенсу.

1.2 Аналітичний розгляд уже існуючих рішень

На сьогоднішній день існує велика кількість різноманітних методів та рішень створення програмного забезпечення. Розробка є складним процесом. Тому, буде правильно спочатку проаналізувати та порівняти ті технології та

програмне забезпечення, яке вже використовують розробники.

В інтернет мережі наявна велика кількість веб-додатків, тому розглянемо декілька їх прикладів.

Карти Google — безкоштовний картографічний веб-сервіс від компанії Google, а також набір застосунків, побудованих на основі цього сервісу й інших технологій Google. Вебсервіс являє собою географічну карту та супутникові знімки всього і надає користувачам можливості панорамного перегляду вулиць, аналізу трафіку у реальному часі, прокладання маршруту. З сервісом інтегрований бізнес-довідник і карта автомобільних доріг, з пошуком маршрутів.

На рисунку 1.1 наведено карти google.

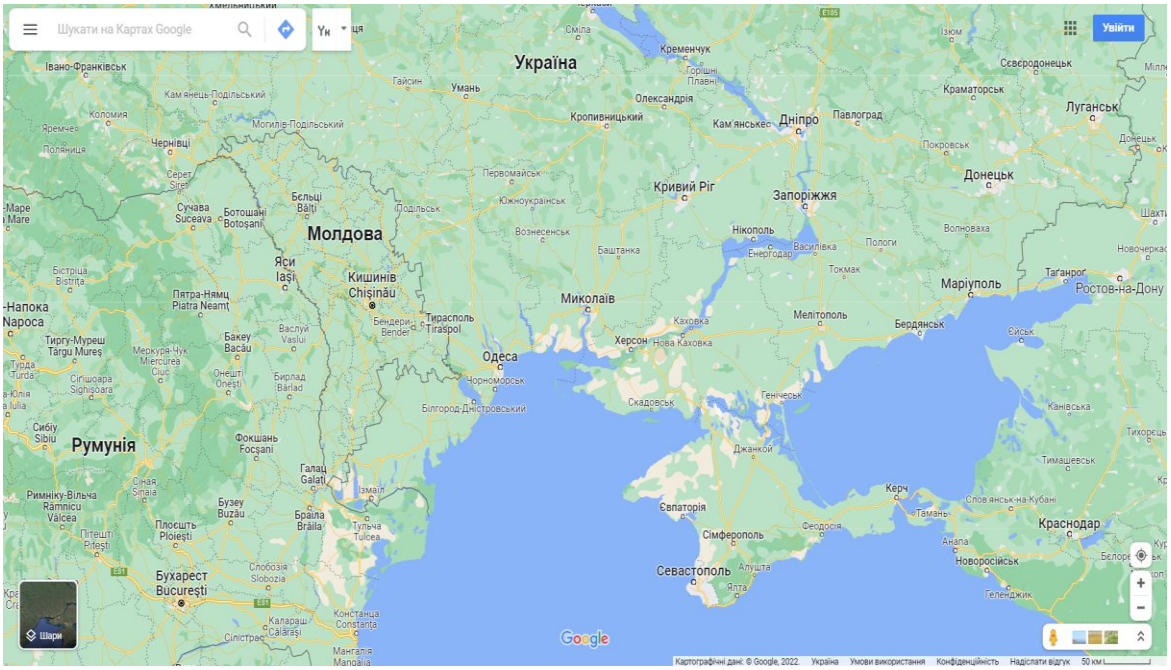


Рисунок 1.1 – Карти Google

Сервіс обміну криптовалют- це онлайн-сервіс, який дозволяє клієнтам обмінювати криптовалюти на інші активи, наприклад, звичайні фіатні гроші або інші цифрові валюти.

Інтернет-нерухомість - це концепція публікації житлових комплексів для продажу чи оренди в Інтернеті, а також для споживачів, бажаючих купити чи орендувати нерухомість через такі платформи.

Можна виділити 5 основних методів розробки вебдодатків:

- ручний - за допомогою HTML;
- за допомогою програмних засобів розробки сайтів;
- за допомогою інструментальних систем таких як CMS;
- з використанням популярних на цей час фреймворків;
- на SaaS-платформах у CLOUD.

При появі стандарту HTML, цей метод був найпоширенішим. Основною програмою для розробки був Notepad. Але у цього методу є істотні недоліки. Цей спосіб досить трудомісткий. І до того ж зробити нормальний Web-сайт без CSS, JavaScript та інших мов програмування досить важко.

Існує багато готових рішень, для більш швидкої і зручної розробки сайтів. Вони надають можливість генерувати html код, розробляти сайт у візуальному режимі і мають багато інших можливостей.

Виділимо декілька інструментальних систем для розробки HTML такі, як програми, що мають у своєму складі візуальні редактори (design-based editor), засоби, які автоматично формують необхідний HTML-код, дозволяючи розробляти Web-сторінки в режимі WYSIWYG, програми-редактори (code-based editors), які надають редактор і допоміжні засоби для автоматизації написання коду.

Розглянемо найбільш популярні design-based редактори:

- Adobe DreamWeaver - один з кращих візуальних редакторів, що генерують HTML код. Він дозволяє працювати в декількох режимах одночасно, з HTML кодом або у візуальному режимі. Але основним недоліком є те, що програма генерує занадто "важкий" код, додаючи багато зайвого. Але, якщо знайомі з HTML, тоді текст HTML можна відредагувати. Ця програмна система випускалася до 2005 року компанією Macromedia, після чого була придбана фірмою Abode.

- Microsoft FrontPage - це простий в засвоєнні і зручний Web-редактор для проектування, підготовки і публікації Web-сайтів. Завдяки інтеграції з сімейством продуктів MS Office, звичного інтерфейсу і великої кількості

шаблонів програма дозволяє швидко засвоїти роботу навіть початківцям, які знайомі з основами роботи в MS Word. При цьому FrontPage не можна назвати рішенням для «чайників»: програма надає широкі функціональні можливості та різноманітні засоби оптимізації при колективній розробці. Вона дозволяє швидко створювати динамічні комплексні Web-вузли практично будь-якої складності.

Розглянемо популярні code-based редактори такі як, Adobe HomeSites - це потужний пакет, до складу якого входить багато корисних функцій і підпрограм. Об'ємний дистрибутив редактора включає в себе, крім самого редактора, редактор TopStyle для редагування таблиць CSS, перевірку орфографії та багато іншого, HotDog - цілком професійний редактор. Вбудована підтримка дуже широкого набору інструментів, що використовуються в Web-дизайні такі, як HTML, CSS, JavaSript, VBScript, ASP, а також DOM - об'єктну модель документа, що використовується при програмуванні на VBScript і JavaScript. При цьому перевірка синтаксису цих інструментів може налаштовуватися в досить широких межах. Наприклад, HTML можна перевіряти на відповідність версії 3.2, 4, або на "перегляді" тільки в Internet Explorer та інше, AceHTML - основні функціональні можливості - подібно HomeSite і FirstPage. З цінних якостей AceHTML треба відзначити вмілу роботу з кодуваннями російської мови. Другий плюс - дуже непогана вбудована утиліта для перегляду графічних файлів у комп'ютері. У просторому вікні відображаються ескізи всіх картинок в директорії, а також їх параметри і розмір у пікселях.

Для створення динамічного сайту можливі два шляхи. По-перше, це написання власних програм, які відповідають за створення потрібних шаблонів і підтримують необхідні функції. При цьому створена система буде повністю відповідати потребам, проте можливо вимагатиме великих програмістських зусиль і часу.

Другий шлях - це скористатися вже існуючими системами, які і називаються системами управління Web-контентом. Перевагою цього шляху є

зменшення витрат часу і сил. До його недоліків можна віднести зниження гнучкості, надання недостатнього або надмірного набору можливостей.

Другий шлях є основним на цей час для створення складних, сучасних сайтів, порталів, Веб-додатків. Це метод з використанням CMS. Вікіпедія дає наступне визначення. CMS це система керування вмістом (контентом) (англ. Content management system, CMS) - комп'ютерна програма чи система, що використовується для забезпечення і організації сумісного процеса створення, редагування та управління текстовими і мультимедійними документами (вмісту чи контента). Звичайно цей вміст розглядається як не структуровані данні предметної задачі в протилежність структурованим даним, що звичайно знаходяться під керуванням СУБД. Звичайно, що встановлення CMS робиться вже на вибраному хостінзі. При цьому як мінімум вимагається FTP доступ та дозвіл роботи MySQL.

Таким чином, відділення дизайну від контенту є головною відміною особливістю динамічних сайтів від статичних. На цій основі можливі подальші удосконалення структури сайту, такі як визначення різних призначених для користувача функцій і автоматизація бізнес-процесів, а саме головне, контроль контенту, що надходить на сайт.

Фреймворк це програмний продукт, який є основою для створення сайтів, але він не має готових рішень для побудови сайтів, не має рішень для виконання певних функцій. Це більш низький рівень ніж CMS. Розробники на фреймворках створюють і інтерфейсну частину, і базу даних, і алгортми та програмні рішення проблемно орієнтованої частини і скоріше не сайту, а веб додатку. Створюючи також його адміністративний інтерфейс.

Фреймворк - це структура програмної системи, що полегшує розробку і об'єднання різних компонентів великого програмного проекту. На відміну від бібліотек, які об'єднують набір підпрограм близької функціональності, фреймворк містить в собі велику кількість різних за призначенням бібліотек. Вживається також слово каркас, а деякі автори використовують його в якості основного, в тому числі не базуючись взагалі на англомовному аналогу.

Можна також говорити про каркасний підхід як про підхід до побудови програм, де будь-яка конфігурація програми будується з двох частин: перша, постійна частина - каркас, незалежний від конфігурації до конфігурації і несе в собі гнізда, в яких розміщується друга, змінна частина - змінні модулі.

Для того щоб відповісти на питання що повинен мати фреймворк, щоб розглядатися як WEB-технологія розглянемо, які види фреймворків є:

- фреймворки програмної системи;
- фреймворки додатків;
- фреймворки концептуальної моделі.

Фреймворк програмної системи - це каркас системи або підсистеми. Він може включати допоміжні програми, мови сценаріїв, все, що полегшує розробку і об'єднання різних компонентів. Від бібліотеки він відрізняється виконанням коду, який написаний для нього, але не виконується сам. До цього виду фреймворків відносяться і фреймворки для WEB.

Фреймворк додатку має стандартну структуру. З ростом необхідності в графічних інтерфейсах користувача з'явилася і необхідність у фреймворках додатків. З їх допомогою простіше створювати засоби для створення графічних інтерфейсів автоматично. Для створення фреймворку додатків використовують об'єктно-орієнтоване програмування. Перший такий Фреймворк написала компанія Apple для Macintosh. Спочатку він був створений за допомогою Паскаль, потім же перероблений в C ++.

Фреймворк концептуальної моделі - це абстрактне поняття даної структури для визначення способів вирішення конкретної проблеми.

WEB фреймворки - це каркас, призначений для створення динамічних веб-сайтів, мережевих додатків, сервісів або ресурсів. Він спрощує розробку і позбавляє від необхідності написання рутинного коду. Багато фреймворків спрощують доступ до баз даних, розробку інтерфейсу, а також зменшують дублювання коду.

Є п'ять типів веб-фреймворків: Request-based, Component-based, Hybrid, Meta and RIA-based.

Request-based: фреймворки, які безпосередньо обробляють вхідні запити. Збереження стану відбувається за рахунок серверних сесій. Приклади: Django, Ruby на Rails, Struts, Grails.

Component-based: фреймворки, які абстрагують обробку запитів всередині стандартних компонентів і самостійно стежать за станом. Своєю поведінкою дані каркаси нагадують стандартні програмні графічні інтерфейси. Приклади: JSF, Tapestry, Wicket.

Hybrid-based: фреймворки, які комбінують Request-based та Component-based фреймворки, беручи під свій контроль всі дані і логічний потік в заснованої на запиті моделі. Розробники мають повний контроль над URL, формами, параметрами, cookies і pathinfos. Однак замість того, щоб відобразити дії і контролери безпосередньо до запиту, гібридні фреймворки забезпечують об'єктну модель компонентів, яка поводиться тотожно в багатьох різних ситуаціях, таких як окремі сторінки, перервані запити, подібні порталу фрагменти сторінок і інтегровані віджети. Компоненти можуть розподілятися окремо і ефективно інтегруватися в інші проекти. Приклади: RIFE.

Meta -based: у фреймворків є ряд базових інтерфейсів для загального обслуговування і основу яка легко розширюється, для інтегрування компонентів і служб. Приклад: Keel.

RIA-based: фреймворки для розробки Rich Internet Applications (RIA). Служать для розробки повноцінних додатків, що запускаються всередині браузера. Приклад: Flex.

Найбільш поширеними є Request-based і Component-based веб-фреймворки. Більшість WEB-фреймворків побудовані на архітектірі MVC.Model View Controller (MVC, «модель- представлення-контролер», «модель-вид- контролер») – схема використання декількох шаблонів проектування, за допомогою яких модель додатки, інтерфейс і взаємодія з користувачем розділені на три окремі компоненти таким чином, щоб модифікація одного з компонентів чинила мінімальний вплив на інші. Дана

схема проектування часто використовується для побудови архітектурного каркаса, коли переходять від теорії до реалізації в конкретній предметної області.

Архітектурний шаблон Модель-Вид-Контролер (MVC) поділяє програму на три частини. У тріаді до обов'язків компоненту Модель (Model) входить зберігання даних і забезпечення інтерфейсу до них. Вигляд (View) відповідальний за представлення цих даних користувачеві. Контролер (Controller) керує компонентами, отримує сигнали у вигляді реакції на дії користувача, і повідомляє про зміни компоненту Модель. Така внутрішня структура в цілому поділяє систему на самостійні частини і розподіляє відповідальність між різними компонентами.

MVC поділяє цю частину системи на три самостійні частини: введення даних, компонент обробки даних і виведення інформації. Модель, як вже було відмічено, інкапсулює ядро даних і основний функціонал з їх обробки. Також компонент Модель не залежить від процесу введення або виведення даних. Компонент виводу Вигляд може мати декілька взаємопов'язаних областей, наприклад, різні таблиці і поля форм, в яких відображається інформація. У функції Контролера входить моніторинг за подіями, що виникають в результаті дій користувача (зміна положення курсора миші, натиснення кнопки або введення даних в текстове поле).

Зареєстровані події транслюються в різні запити, що спрямовуються компонентам Моделі або об'єктам, відповідальним за відображення даних. Відокремлення моделі від вигляду даних дозволяє незалежно використовувати різні компоненти для відображення інформації. Таким чином, якщо користувач через Контролер внесе зміни до Моделі даних, то інформація, подана одним або декількома візуальними компонентами, буде автоматично відкоригована відповідно до змін, що відбулися на SaaS-платформах у Cloud.

SaaS-платформи для створення сайтів - це можливість запустити досить простий веб-проект дуже швидко і досить дешево (ще і на умовах оренди і без необхідності хостінгу). Щось подібне до Гугл сайтів, але зі значно більшими

можливостями. Рішення підходить для простих сайтів, тимчасових проектів і для перевірки бізнес-ідей. SaaS-платформи, як і CMS, бувають специфічними (наприклад, тільки для інтернет-магазинів) і універсальними (для всіх типових видів сайтів). Цей метод зараз активно розвивається та використовується.

Які проекти варто реалізовувати на SaaS-платформах?

SaaS-рішення мають свої переваги і недоліки, тому не кожен проект може бути реалізований подібним способом. На SaaS має сенс створювати прості сайти, які не особливо вимогливі до дизайну і ні відразу, ні в перспективі не зажадають яких-небудь доопрацювань логіки роботи та навігації. SaaS відмінно підходить для запуску проектів, мета яких полягає в перевірці на практиці бізнес-гіпотез. Можна і для створення простих проектів, вимога до яких одне - «потрібен сайт», а також для створення «заглушок» при розробці серйозних проектів. Хоча простий сайт доцільно створювати, наприклад, за допомогою засобів Гугл сайт.

Кроки створення сайтів на SaaS-платформах:

- крок 1 Зареєструватися;

- крок 2 Вибрати тип сайту (наприклад, сайт-візитка, інформативний сайт або інтернет-магазин);

- крок 3 Вибрати потрібні модулі (новини, каталог, форум та інше);

- крок 4 Вибрати шаблон дизайну зі переліку шаблонів;

- крок 5 Ввести необхідну інформацію в систему управління контентом;

- крок 6 Ввести контент.

SaaS-рішення входить все необхідне для повноцінної роботи проекту - не треба вибирати, встановлювати і налаштовувати CMS, не треба замовляти хостинг і налаштовувати сервер, а в подальшому не треба займатися технічним супроводом проекту. Все це робить за вас SaaS-рішення.

Мінуси SaaS-платформ для створення сайту:

- фактично це шаблонний дизайн - оформлення сайтів на SaaS-платформах проводиться за готовими шаблонами (часто не дуже високої якості), які можна тільки «розфарбувати» і на деяких платформах можна

окремі блоки місцями поміняти. Для сайтів, до яких пред'являються вимоги до оформлення, такі рішення не підходять.

- рамки функціональних можливостей - якщо платформа «не має якихось можливостей», то це ніяк не виправити. Програмний продукт типовий і його налаштування під індивідуальні побажання обмежена. Якщо проект відразу або в перспективі повинен вирішувати специфічні завдання і гнучко налаштовуватися, то SaaS-платформа для його розробки не підходить.

Приклади SaaS-платформ:UMI, WIX, InSales, Shopify, Setup, uCoz - деякі з цих платформ специфічні (тільки для простих сайтів або тільки для інтернет-магазинів), а деякі - досить універсальні.

Якщо проект не має ніяких істотних вимог до дизайну і до функціональності, то має сенс звернути увагу на ці SaaS-рішення. В іншому випадку, вибирайте іншу платформу для створення сайтів.

Розділ 2 Формування технічного завдання

2.1 Постановка задачі та призначення

У зв’язку з активним розвитком міського електротранспорту, зокрема сервісів оренди електросамокатів, виникає нагальна потреба у створенні зручних та ефективних засобів для моніторингу, керування і аналітики такого транспорту в межах населених пунктів. Особливо це стосується великих міст, де кількість пристроїв постійно зростає, а контроль за їх використанням, справністю, місцем розташування та відповідністю до встановлених правил є критично важливим.

Метою дипломного проєкту є розробка та впровадження сучасного вебзастосунку, системи моніторингу електросамокатів, яка дозволить візуалізувати у реальному часі місцезнаходження електросамокатів на інтерактивній карті, забезпечити зручний доступ до інформації про технічний стан пристроїв та їхній статус, а також надати інструменти для гнучкого керування парком самокатів.

Система повинна взаємодіяти з локальним API, який надає інформацію про всі пристрої, їхнє поточне положення, рівень заряду акумулятора, модель, а також журнал подій — історію переміщень, повідомлення про технічні несправності або порушення правил (наприклад, виїзд за межі дозволеної зони).

Щоб оцінити трудомісткість створення програмного забезпечення, потрібно поділити процес розробки на окремі етапи. Для кожного з них слід провести аналіз і надати оцінку, базуючись на фактичному часі, витраченому на виконання відповідних завдань. У таблиці 5.1 представлено ключові стадії, які проходить розробка програмного продукту.

Таблиця 2.1 – Етапи розробки програмного забезпечення

Етап	Вид виконуваних робіт
1	Обґрунтування необхідності розробки системи
2	Пошук та аналіз існуючих методів та рішень

3	Вибір та обгрунтування технологій
4	Проєктування системи
5	Кодування
6	Тестування
7	Впровадження програмного забезпечення
8	Написання документації

Розрахунок тривалості кожного етапу проводиться в таблиці 2.2

Таблиця 2.2 – Розрахунок тривалості робіт

Етап	Мінімальна кількість, днів	Максимальна кількість, днів	Фактична кількість, днів
1	1	3	1
2	3	6	4
3	3	7	6
4	11	21	13
5	11	21	18
6	2	4	3
7	2	4	2
8	2	4	3
Всього	35	70	50

Основні задачі, які ставляться в межах реалізації проєкту - розробити вебінтерфейс для адміністративного та аналітичного використання, що дозволить (з дійснювати пошук електросамокатів за заданими параметрами (модель, заряд, статус, місце розташування), фільтрувати дані в реальному часі, переглядати детальну інформацію по кожному самокату), інтегрувати google maps api для відображення розташування електросамокатів на мапі з можливістю (оновлення координат у режимі реального часу, кластеризації пристроїв у разі великої кількості на мапі, масштабування та взаємодії з конкретним об’єктом (наприклад, при натисканні на маркер відкривати інформаційну панель пристрою)), реалізувати візуалізацію зон покриття (зон дозволеного пересування, зон, де самокат не повинен перебувати (наприклад, приватні території, парки, пішохідні зони), відображення порушень при виїзді

за межі встановленої території), забезпечити можливість перегляду історії пересування окремих самокатів за певний період часу. Це дозволить аналізувати маршрути користувачів, виявляти підозрілі переміщення або несанкціоноване використання пристроїв, забезпечити масштабовану архітектуру вебзастосунку на основі JavaScript-фреймворку Adonis.js, що дозволить легко розширювати систему в майбутньому та інтегрувати додаткові модулі, наприклад аналітику використання, інтеграцію з зовнішніми сервісами (наприклад, погодні API для підвищення точності даних про умови експлуатації).

Розроблений вебдодаток призначений для операторів сервісів прокату електросамокатів, які отримають інструмент для щоденного контролю і керування своїм парком техніки, міських адміністрацій, які зможуть використовувати систему як засіб контролю за використанням електросамокатів на території населеного пункту, технічного персоналу, який зможе оперативно отримувати інформацію про несправності або потребу в обслуговуванні окремих пристроїв.

Таким чином, реалізація даного вебзастосунку дозволить оптимізувати роботу сервісів мікромобільності, покращити взаємодію між користувачами і операторами, а також підвищити безпеку, ефективність і прозорість використання електросамокатів у міських умовах.

2.2 Вимоги до додатку

Перед початком проєктування та розробки веб-додатку необхідно чітко сформулювати функціональні та нефункціональні вимоги до системи, які визначають основні задачі, поведінку додатку, а також технічні характеристики, що впливають на його ефективність, надійність та зручність використання.

Система повинна забезпечити повноцінну підтримку моніторингу та керування електросамокатами в реальному часі, з можливістю інтерактивної взаємодії з користувачем. Додаток повинен бути адаптований до сучасних веб-

браузерів, мати адаптивний інтерфейс для різних типів пристроїв (настільні ПК, ноутбуки, планшети).

Додаток реалізується за допомогою JavaScript-фреймворку Adonis.js на серверній частині та з використанням Node.js бібліотек для клієнтської частини.

Функціональні вимоги визначають, що саме повинен уміти додаток. Для системи моніторингу електросамокатів передбачено наступний функціонал

Можливість здійснювати пошук самокатів за:

- унікальним ідентифікатором або серійним номером;
- моделлю;
- статусом (активний, неактивний, несправний, заряджається тощо);
- рівнем заряду акумулятора;
- місцем розташування (місто, вулиця, координати);
- часовими діапазонами (наприклад, останній раз активний сьогодні/вчора тощо).

Візуалізація на карті:

- відображення всіх активних самокатів на карті у режимі реального часу;
- підтримки масштабування та переміщення мапи;
- кластеризації маркерів при великій кількості пристроїв;
- відображення інформаційних карток із детальною інформацією про самокат при натисканні на маркер.

Відображення детальної інформації про пристрої:

- модель;
- серійний номер;
- рівень заряду акумулятора (у відсотках);
- поточний статус (активний, несправний, заряджається);
- поточне місце розташування (координати, адреса);
- остання активність (час і дата).

Визначення зон покриття:

- дозволеного пересування;
- заборонених зон (наприклад, приватна територія або пішохідна зона);
- зон паркування.

Нефункціональні вимоги - вебдодаток повинен швидко реагувати на дії користувача та оновлювати дані в режимі реального часу, ч ас відгуку не повинен перевищувати 1–2 секунд при стандартних запитах (пошук, фільтрація). Надійність - додаток повинен коректно працювати при нестабільному інтернет-з'єднанні (наприклад, кешування попередніх даних), п овинна бути реалізована система резервного копіювання бази даних. Масштабованість – архітектура повинна дозволяти масштабування системи при збільшенні кількості пристроїв або регіонів використання. Безпека - авторизація і автентифікація для адміністративного доступу, захист API-запитів (JWT-токени, rate limiting), валідація вхідних даних для запобігання SQL-ін'єкціям і XSS-атакам. Адаптивність - інтерфейс повинен бути адаптованим під різні пристрої (десктоп, планшет, смартфон), підтримка сучасних браузерів (Chrome, Firefox, Safari, Edge).

Таким чином, сформульовані вимоги забезпечують всебічне охоплення функціоналу системи моніторингу електросамокатів і створюють чітке технічне завдання для подальшої розробки вебдодатку. Вони дозволяють побудувати ефективну, зручну та безпечну систему, яка задовольняє потреби як операторів сервісів, так і міських органів управління.

2.3 Опис вхідної інформації

У рамках реалізації вебдодатку для моніторингу та управління електросамокатами важливою складовою є коректне визначення та обробка вхідної інформації. Вхідна інформація - це сукупність даних, які надходять від користувача або інших джерел (API, баз даних, сенсорних систем) і є основою для функціонування всіх модулів системи.

Одним із джерел вхідної інформації є введення текстових запитів безпосередньо користувачем через інтерфейс пошуку або панель фільтрації. Такі дані включають текстові ключові слова або ідентифікатори, які вводяться з клавіатури для пошуку конкретного самоката (наприклад, номер моделі, серійний номер, внутрішній ID), значення параметрів фільтрації, зокрема статус пристрою (активний, неактивний, заряджається, несправний), модель (назва моделі або виробника), рівень заряду акумулятора (у відсотках або умовах «високий / середній / низький»), географічні критерії (місто, район, адреса) і вся інформація яка вводиться з клавіатури.

2.4 Опис результуючої інформації

Результуюча інформація є підсумком обробки вхідних даних у вебзастосунку та відображається користувачу у зручній для сприйняття формі. Вона є ключовим елементом взаємодії користувача з системою, оскільки забезпечує зворотний зв’язок, інформування про стан пристроїв, ефективність їх використання та загальну аналітику парку електросамокатів.

У межах вебдодатку результуюча інформація представлена у таких формах.

Візуальна інформація на інтерфейсі:

- Карта з маркерами електросамокатів - відображає місцезнаходження кожного пристрою в реальному часі. Колір або іконка маркера змінюється залежно від статусу (наприклад, зелений - активний, червоний - несправний, жовтий - низький заряд);
- Інформаційні картки при натисканні на маркер - містять короткий опис: ID самоката, модель, рівень заряду, час останнього оновлення, статус;
- Вікна повідомлень та сповіщень - наприклад, при виїзді самоката за межі дозволеної зони, або при зниженні заряду нижче допустимого рівня;

- Форми результатів пошуку та фільтрації - список пристроїв, які відповідають введеним параметрам, із можливістю перегляду на мапі.

Аналітична інформація:

- Графіки змін заряду акумулятора окремого самоката протягом обраного періоду;
- Історія переміщень - лінії маршруту самоката на мапі за певний час;
- Статистика по парку самокатів - кількість активних, заряджених, несправних пристроїв, середній час використання, середній пробіг тощо.

Навігаційна інформація

- Динамічний інтерфейс сторінок - після переходу користувача між вкладками або виконання дій (наприклад, натискання кнопки «Деталі») завантажуються відповідні сторінки з контентом;
- Посилання на сторінки власників або адміністраторів - наприклад, при натисканні на самокат, система може запропонувати перейти на сторінку відповідального оператора або власника.

Вебдодаток реалізований у формі інтерактивного інтерфейсу, що дозволяє розглядати кожну взаємодію користувача (пошук, натискання, вибір фільтрів) як запит, а результуючу інформацію - як відповідь системи на цей запит. Введення моделі самоката, результат: перелік усіх пристроїв такої моделі на мапі. Обрання фільтра «заряд < 20%», результат: відображення в таблиці лише тих самокатів, які потребують зарядки.

Результуюча інформація формується динамічно на основі даних, отриманих від API. Вона не є статичною, а оновлюється залежно від вибору користувача. Повідомлення, кнопки, графіки та таблиці заздалегідь описані в коді інтерфейсу, а їх вміст заповнюється залежно від API-відповіді. Це забезпечує адаптивність системи до змін, зручність та гнучкість у використанні.

Результуюча інформація у вебзастосунку для моніторингу електросамокатів є важливим засобом інформування користувача про поточний стан системи, ефективність роботи парку та прийняття управлінських рішень. Вона представлена у різних формах - візуальних, текстових, графічних - і формується в залежності від дій користувача, дозволяючи ефективно керувати великим обсягом телеметричних та геолокаційних даних.

2.5 Етапи розробки

Розробка програмного забезпечення - це комплексний процес, який складається з послідовних етапів, кожен з яких відіграє важливу роль у створенні якісного і надійного продукту. Для реалізації дипломного проєкту «Система моніторингу та управління електросамокатами» була застосована класична модель життєвого циклу програмного забезпечення.

Першим етапом розробки є аналіз проблемної області та збір вхідної інформації про майбутню систему. На цьому етапі досліджується специфіка предметної області (моніторинг електросамокатів), цільова аудиторія (компанії-орендодавці, муніципальні структури), аналізуються наявні рішення та аналоги на ринку, виявляються основні проблеми, які система має вирішити.

Також формується загальна ідея системи, описується її призначення, оцінюються технології, які доцільно використовувати для реалізації - зокрема, було обрано JavaScript та фреймворк Adonis.js, а також інтеграцію з Google Maps API.

На цьому етапі створюється концептуальна модель системи. Визначаються основні функціональні вимоги (пошук, фільтрація, візуалізація, сповіщення), нефункціональні вимоги (швидкодія, безпека, зручність інтерфейсу), сценарії взаємодії користувача з системою (User Stories, Use Cases), тип додатку (вебдодаток з клієнт-серверною архітектурою). Також

проводиться поділ на модулі - наприклад: модуль мапи, модуль аналітики, модуль сповіщень, модуль адміністративної панелі.

Проєктування - це критично важливий етап, на якому здійснюється деталізація архітектурних рішень. Тут визначається логічна структура системи, зв’язки між модулями, способи зберігання та обробки даних (база даних, формат API-відповідей), створюються діаграми класів, послідовності, станів (UML).

Проєктування дозволяє розробнику візуалізувати, як саме працюватиме система, які компоненти потрібні, як вони взаємодіятимуть між собою, і які ресурси будуть задіяні.

На етапі реалізація виконується безпосереднє написання коду. Кожен з модулів реалізується відповідно до спроєктованої архітектури. Бекенд реалізовано за допомогою Adonis.js, з побудовою RESTful API. Фронтенд використовує сучасні JavaScript бібліотеки та компоненти для візуалізації даних на карті. Інтеграція з Google Maps API дозволяє виводити електросамокати у вигляді маркерів, динамічно оновлювати їхній стан і розташування.

Після реалізації базової функціональності проводиться комплексне тестування застосунку, яке включає функціональне тестування (правильність виконання команд), навантажувальне тестування (перевірка поведінки при великій кількості запитів), тестування інтерфейсу (відображення елементів у різних браузерах), перевірка на коректність обробки помилок (некоректний ввід, відсутність мережі тощо).

Після завершення тестування вебдодаток готується до публічного використання. На цьому етапі проєкт розгортається на сервері (використовується хостинг або хмарні сервіси), проводиться налаштування середовища виконання (база даних, API), створюються інструкції для кінцевого користувача, адміністраторів та технічних спеціалістів, документується архітектура, конфігурації, API-ендпоінти. Це дозволяє іншим розробникам легко супроводжувати та масштабувати систему в майбутньому.

Супровід - це етап після впровадження, що включає виправлення помилок, виявлених під час реального використання, оптимізацію продуктивності та ресурсоємності, адаптацію системи до нових умов (оновлення API, додавання нових моделей самокатів), додавання нової функціональності за вимогами замовника. Таким чином, життєвий цикл програмного продукту не закінчується на момент запуску, а триває шляхом його постійного вдосконалення.

Процес розробки програмного забезпечення передбачає ретельне планування, чітке структурування етапів та послідовне втілення ідей у вигляді коду. Дотримання всіх етапів розробки дозволяє досягти стабільної, надійної, функціонально завершеної системи, що відповідає вимогам користувача та здатна адаптуватися до нових умов.

Розділ 3 Проектування схеми роботи

3.1 Архітектура програмного продукту

У процесі розробки системи моніторингу та управління електросамокатами було прийнято рішення використовувати клієнт-серверну архітектуру, яка на сьогодні є найпоширенішою концепцією побудови сучасних вебдодатків. Такий підхід дозволяє забезпечити масштабованість, безпеку, повторне використання коду та зручність у подальшій підтримці й розвитку системи.

Клієнт-серверна архітектура передбачає наявність двох основних сторін:

- Клієнт - програмний компонент (веббраузер, мобільний застосунок тощо), який надсилає запити до сервера та виводить отримані дані користувачу.
- Сервер — програмна частина, що приймає запити клієнта, обробляє їх, взаємодіє з базою даних та іншими сервісами, а також надсилає відповіді.

Архітектура клієнт-серверної системи:

- користувач - працює з веб-інтерфейсом;
- інтерфейс користувача (UI) - приймає введення від користувача і відображає інформацію;
- API-контролери - обробляють HTTP-запити від інтерфейсу;
- сервісна логіка - тут реалізована основна логіка застосунку (перевірки, бізнес-правила);
- репозиторій - доступ до бази даних, ізоляція запитів;
- база даних - зберігає всю інформацію про самокати, користувачів, зони тощо.

На рисунку 3.1 наведена блок схема клієнт-серверної системи.

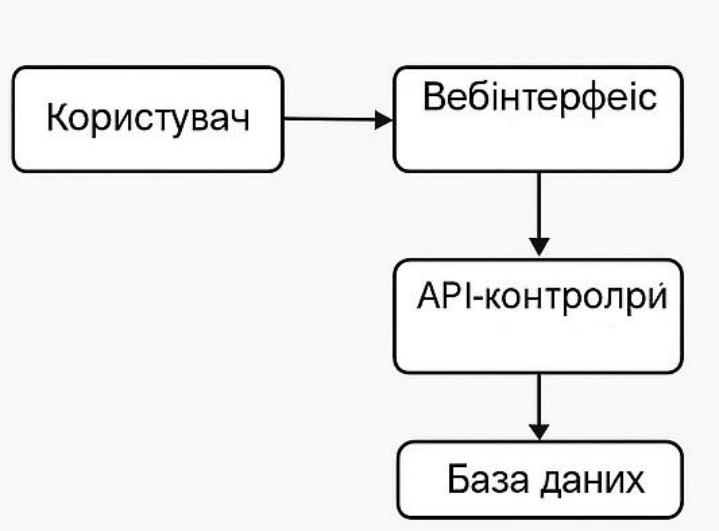


Рисунок 3.1 – Блок-схема клієнт-серверної системи

У межах цього проєкту серверна частина реалізована на фреймворку AdonisJS, що базується на Node.js, і забезпечує RESTful API для взаємодії з клієнтом. Сервер може бути розгорнутий як на локальному середовищі розробника, так і на хмарному сервері, наприклад, за допомогою DigitalOcean, Heroku, Vercel або AWS.

Для ефективної організації коду та забезпечення гнучкості при розробці та розширенні функціональності, застосовано багаторівневу (багатошарову) архітектуру, яка складається з таких основних компонентів.

Модель - це представлення сутностей предметної області (наприклад: самокат, зона, користувач, маршрут, сповіщення). У середовищі AdonisJS це класи, які відповідають таблицям у базі даних та описують структуру даних.

Модель містить тільки структуру даних, інколи включає базову валідацію або взаємозв’язки між об’єктами (наприклад, hasMany, belongsTo), не містить логіки обробки або взаємодії з інтерфейсом. Це дозволяє дотримуватися принципу розділення відповідальностей (Separation of Concerns).

Сервісний шар реалізує бізнес-логіку застосунку, тобто саме тут відбувається обробка даних згідно з вимогами проєкту:

- визначення дозволених зон пересування самокатів,
- формування списку активних пристроїв,
- застосування правил фільтрації або сортування,

- розрахунок відстаней, часу, обробка подій тощо.

Кожен сервіс - це автономний клас або модуль, який працює з моделями та репозиторіями, реалізуючи логіку, орієнтовану на задачі користувача або адміністратора.

Репозиторій відповідає за взаємодію з базою даних:

- виконання CRUD-операцій (створення, читання, оновлення, видалення),
- формування складних SQL-запитів або використання ORM-запитів,
- інкапсуляція способу доступу до даних.

Це дозволяє змінювати спосіб зберігання або джерело даних (наприклад, перехід з реляційної БД на NoSQL) без впливу на бізнес-логіку або інші частини коду.

Представлення або View/UI - це компонент, з яким безпосередньо взаємодіє користувач. У цьому проєкті роль представлення виконує вебінтерфейс, що працює через браузер і надсилає запити до серверного API, відображає інформацію на карті за допомогою Google Maps API, надає зручні засоби для фільтрації, перегляду, пошуку, керування пристроями.

Представлення може бути побудовано за допомогою фреймворків React, Vue або стандартних HTML/JS/Bootstrap технологій. Воно умовно поділяється на візуальний елемент - відповідальний за відображення (UI), керуючий елемент - обробляє події, взаємодіє з API, формує дії користувача.

Обрана багатошарова клієнт-серверна архітектура має низку суттєвих переваг:

- масштабованість - серверна частина може обслуговувати багато клієнтів одночасно;
- гнучкість - легко змінювати логіку на одному рівні, не зачіпаючи інші;
- повторне використання коду - сервіси та репозиторії можна застосовувати у різних компонентах;
- тестованість - можна окремо тестувати сервіси, моделі, API,

інтерфейс;

- можливість розподілу ролей - бекенд, фронтенд, DevOps - незалежні один від одного.

Таким чином, клієнт-серверна багатошарова архітектура, що використовується у даному проєкті, дозволяє досягти високої структурованості, прозорості та розширюваності коду. Завдяки чіткому розподілу обов'язків між компонентами, систему легко підтримувати, масштабувати та модифікувати відповідно до нових бізнес-вимог.

3.2 Створення UML-діаграм

У процесі розробки програмного продукту одним з ключових етапів є моделювання. Для цього було використано UML (Unified Modeling Language) - уніфіковану мову моделювання, яка є стандартом у сфері об'єктно-орієнтованого проєктування. UML дозволяє створювати абстрактну модель системи з використанням графічних нотацій, що сприяє кращому розумінню, комунікації в команді та документації процесів.

UML є незалежним від мови програмування засобом опису систем і застосовується на всіх етапах життєвого циклу ПЗ: від аналізу вимог до проєктування, реалізації та супроводу. Його застосування значно підвищує ефективність командної роботи, дозволяє швидше виявляти та усувати помилки на ранніх етапах проєктування.

Переваги використання UML:

- полегшення комунікації між розробниками, аналітиками та зацікавленими сторонами;
- можливість створення структурованого, візуального подання системи;
- підвищення якості кінцевого продукту за рахунок кращого планування;
- автоматична генерація частин коду з діаграм;
- формалізація вимог до системи.

Діаграма прецедентів дозволяє визначити зовнішніх учасників системи (акторів) та дії (прецеденти), які вони можуть виконувати у межах взаємодії із системою. Це базова діаграма, що використовується на етапі збору вимог. Вона дозволяє побачити, які функціональні можливості реалізовує система та хто з ними взаємодіє.

На діаграмі ДП.00.000.00.000.Д3 зображено основні прецеденти взаємодії користувачів (адміністраторів, клієнтів, системних сервісів) із системою, включаючи операції реєстрації, входу, створення та перегляду об'єктів, а також керування блокуванням та оплатами.

Діаграма послідовності демонструє часовий порядок взаємодії між об'єктами системи під час виконання конкретного сценарію. Вона дозволяє відстежити логіку викликів функцій, відповідей, умовних переходів. Це важливо для розуміння динаміки системи.

На діаграмі ДП.0000.0000.00.000.Д1 відображено послідовність дій у межах реалізації CRUD-функціональності через API, що реалізується згідно з принципами DDD (Domain Driven Design) та структурою Onion Architecture, яка забезпечує ізоляцію доменної логіки від інфраструктури.

Діаграма активності дозволяє описати логіку поведінки системи або окремої функції у вигляді потоку дій. Вона подібна до блок-схеми та використовується для візуалізації алгоритмів і бізнес-процесів.

На діаграмі ДП.0000.00000.00.000.Д2 представлено алгоритм обробки запитів користувача, включаючи авторизацію, перевірку даних, обробку транзакцій і повернення результатів. Такий підхід дає змогу оптимізувати процеси та виявити неефективні вузли системи.

Діаграма розгортання використовується для представлення фізичного розміщення компонентів програмного забезпечення на обчислювальних

вузлах. Вона показує, як і де розгортаються сервіси, які взаємодіють один з одним.

На діаграмі ДП.0000.00000.00.000.Д5 проілюстровано розгортання системи у вигляді клієнт-серверної архітектури, де серверна частина (API, база даних, обробники подій) розгорнута на хмарному середовищі, а клієнтська частина — на пристроях користувачів. Також показано взаємодію з мережею блокчейну.

На окремій діаграмі ДП.0000.00000.00.000.Д4 зображено механізм синхронізації із мережею блокчейн. Представлено ключові компоненти, операції над блоками та транзакціями, що ілюструє інтеграцію централізованої системи з децентралізованими обчисленнями.

UML-діаграми виконують роль «архітектурного креслення» програмного продукту. Їх використання дозволило чітко окреслити межі системи, визначити ролі користувачів і їхні сценарії взаємодії, побудувати логічні послідовності викликів функцій та їхню реалізацію, візуалізувати розгортання інфраструктури та компоненти, здійснити проєктування модулів, адаптованих до розширення й підтримки.

Цей підхід значно спростив реалізацію програмного продукту, уніфікував бачення серед команди та забезпечив прозорість у процесах розробки.

3.3 Проектування інтерфейсу

Інтерфейс користувача (UI) виступає центральною точкою взаємодії між людиною та інформаційною системою. Для веб-додатку “Система моніторингу електросамокатів” інтерфейс має не лише відображати великий обсяг телеметричних даних, а й дозволяти адміністратору швидко реагувати на події (низький заряд, вихід із зони, несправність). Нижче подано деталізований опис принципів, макетів і технічних рішень, що лягли в основу проєктування UI.

Принципи і вимоги до UI:

- Пізнавальна простота (один головний екран із картою; всі другорядні функції (фільтри, деталі, статистика) відкриваються у бічних панелях або модальних вікнах);
- Мінімізація дій - типові сценарії (знайти самокат, відкрити картку, змінити статус) виконуються за ≤ 3 клаци;
- Відгук у реальному часі (webSocket-канал надсилає пуш-події; UI миттєво оновлює маркери, лічильники, сповіщення);
- Адаптивність (сітка Tailwind + CSS Grid забезпечує комфортне відображення від 320 px до ультраширокоих моніторів; бічна панель ховається на мобільних).

Інформаційна структура екранів:

- Toolbar (логотип, індикатор підключення, глобальний пошук ID/моделі);
- Карта Google Maps;
- Бічна панель-фільтр;
- Адмін-розділ (CRUD для користувачів і ролей (Formik + Zod валідація), JSON-редактор полігонів зон (react-leaflet-draw), планувальник обслуговування (Drag & Drop календар)).

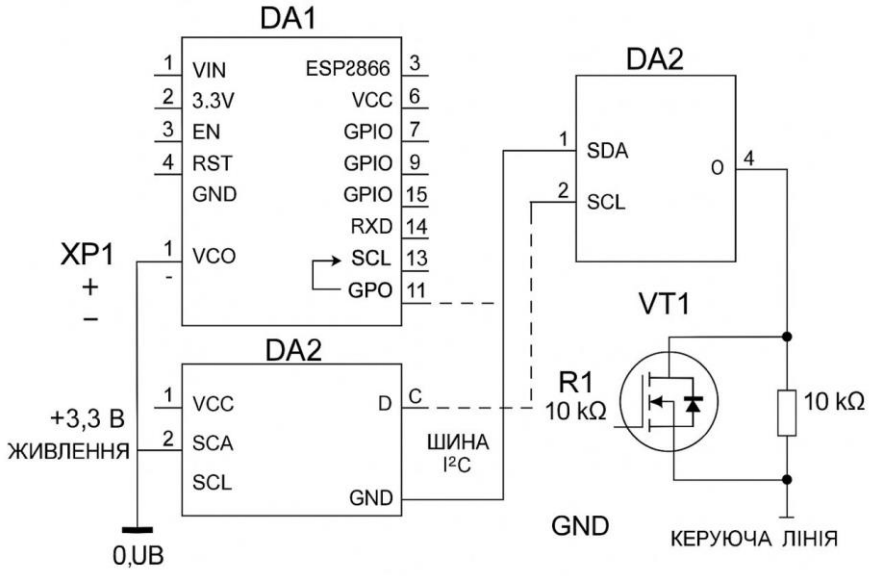
Робота інтерфейсу по створенні зони адміністратором наведено на блок-схемі 3.2.

Пояснення до блок-схеми:

- Адміністратор відкриває мапу - вебінтерфейс відображає інтерактивну карту;
- кнопка «Створити зону» - ініціює режим малювання полігону;
- малювання зони на мапі - адміністратор виділяє межі зони;
- введення даних - назва зони, колір, опис;
- надсилання даних - через API запит у бекенд;
- збереження у БД - сервер обробляє і зберігає інформацію;
- підтвердження - зона успішно створена, оновлюється інтерфейс.

Рисунок 3.2 – Блок-схема по створенні зони

Спроєктований інтерфейс поєднує картографічну візуалізацію, оперативні сповіщення та аналітичні панелі, зберігаючи при цьому високу доступність і продуктивність. Завдяки чіткій інформаційній архітектурі, модульним компонентам і системі дизайну на Tailwind, UI легко підтримувати та масштабувати разом із бізнес-вимогами сервісу моніторингу електросамокатів.



Рисонок 3.3- Принципова схема

Розділ 4 Програмна реалізація проекту тестування та укспуатація

4.1 Програмна реалізація проекту

У процесі створення програмного забезпечення для дипломного проєкту було використано сучасні підходи до розробки серверних застосунків, з дотриманням принципів масштабованості, модульності та підтримуваності коду.

Основною мовою програмування, яка була використана в дипломному проєкті, стала TypeScript — типізоване надбудування над JavaScript, яке дозволяє створювати більш надійний, масштабований і зручний у підтримці код. Сильна типізація TypeScript значно спрощує налагодження коду та зменшує кількість помилок, а також підвищує читабельність логіки застосунку.

Переваги TypeScript у контексті реалізації проєкту:

- підтримка об’єктно-орієнтованого програмування (ООП);
- висока інтеграція з IDE (наприклад, VS Code);
- спрощення роботи з великими структурами даних;
- зручна робота з API, DTO, типами.

Основним фреймворком виступив AdonisJS 5 — сучасний серверний фреймворк для Node.js, що базується на TypeScript. AdonisJS є потужним інструментом, який надає зручну архітектуру MVC (Model-View-Controller), систему маршрутизації, ORM Lucid, вбудовану підтримку валідації, автентифікації, сесій та багато іншого «з коробки». Це дозволило сфокусуватися саме на бізнес-логіці проєкту, а не на низькорівневій реалізації типових задач веб-розробки.

Для структуризації коду в рамках проєкту було застосовано підхід DDD (Domain-Driven Design) — предметно-орієнтовану розробку. Основна ідея полягає у чіткому розділенні відповідальностей у проєкті за предметними областями.

Крім того, враховуючи об'єктно-орієнтовану природу AdonisJS та TypeScript, реалізація програмного коду базувалася на SOLID-принципах, що дозволило зробити рішення максимально масштабованим, стабільним та підтримуваним:

- S - Single Responsibility Principle: кожен сервіс, контролер чи клас відповідає лише за одну частину логіки (наприклад, обробка запиту користувача, доступ до БД, тощо);
- O - Open/Closed Principle: більшість сервісів реалізовані так, що можуть бути доповнені через спадкування або делегування, не змінюючи вихідного коду;
- L - Liskov Substitution Principle: в проекті використовувались інтерфейси, що дозволяють легко підставляти реалізації без зміни логіки;
- I - Interface Segregation Principle: інтерфейси розділені на вузькоспеціалізовані, що дозволяє сервісам реалізовувати лише ті частини логіки, які їм дійсно потрібні;
- D - Dependency Inversion Principle: впровадження залежностей реалізовано через сервіс-контейнер AdonisJS, що забезпечує гнучкість та тестованість коду;

Програмна структура проєкту побудована у вигляді модулів, які відповідають за певні функці. Config - зберігання конфігурацій проєкту: API ключі, таймаути, параметри підключення до сторонніх сервісів. Domains - доменні області, що реалізують бізнес-логіку системи. Наприклад: User, Post, Subscriber, Block, Telegram, MediaUpload, тощо. Http (Controllers, Middleware) - модулі, що відповідають за обробку HTTP-запитів, авторизацію, валідацію вхідних даних. Validators - класи, що відповідають за перевірку запитів (на основі Validator з AdonisJS). Services / UseCases - окремі сервіси або сценарії використання (Use Cases), що інкапсулюють прикладну логіку. Providers - налаштування та ініціалізація зовнішніх бібліотек, залежностей, кастомних сервісів (наприклад, TelegramBotProvider, EmailServiceProvider). Database

(Models, Migrations, Seeders) - опис моделі даних через ORM Lucid: визначення структур таблиць, зв’язків між ними, а також наповнення тестовими даними.

В якості СУБД використовувалась PostgreSQL. Всі моделі реалізовані з використанням ORM Lucid, яка підтримує повноцінну роботу з реляційними зв’язками (hasOne, hasMany, belongsTo) та дозволяє будувати складні запити на основі бізнес-логіки.

Завдяки обраному стеку технологій - TypeScript + AdonisJS 5, а також підходам DDD та SOLID - вдалося реалізувати стабільну, логічно розділену, зручну у супроводі систему з чіткою архітектурою, яка відповідає сучасним стандартам веб-розробки.

4.2 Тестування проекту

Тестування програмного забезпечення – це процес, що використовується для виміру якості розроблюваного проєкту. Тестування пронизує весь життєвий цикл ПЗ. Це ітераційний процес. До цього процесу входить виконання програми з метою пошуку помилок. Це реалізовується за допомогою введення наперед продуманих даних, які можуть викликати збої і спостереження за тим, як поводить себе програма при такому ході подій.

Модульне тестування (unit) — це метод тестування , який полягає в окремому тестуванні кожного модуля коду програми. Модулем називають найменшу частину програми, яка може бути протестованою. Зазвичай unit-тести застосовують для того, щоб упевнитися, що код відповідає вимогам архітектури та має очікувану поведінку.

Інтеграційне тестування — це фаза тестування програмного забезпечення, під час якої окремі модулі програми комбінуються та тестуються разом, у взаємодії. Інтеграційне тестування виконується після модульного тестування та перед верифікацією та валідацією ПЗ. Якщо розглядати цей процес як систему, то на вхід їй подаються модулі, які вже пройшли модульне тестування; потім модулі групуються в більші частини,

виконуються тести передбачені планом, а на виході системи — інтегрована система, що готова до системного тестування.

Системне тестування є одним з рівнів тестування програмного забезпечення. Системне тестування тестує інтегровану систему для перевірки відповідності всім вимогам. Перевірка повноти та правильності документації користувача є важливою частиною системного тестування. Всі тестові комбінації повинні розроблятися тільки з використанням документації користувача.

В решті, ручне тестування здійснюється за допомогою натискання кнопок і перевірки результатів їх відправлювання в самому інтерфейсі веб-сторінки.

Отже, процес тестування відіграє величезну роль у реалізації готового проєкту, щоб запобігти помилок в процесі експлуатації.

4.3 Інструкція з інстаціляції проекту

Для інсталяції проєкту спочатку потрібно переконатися в тому, що пристрій буде підтримувати дане програмне забезпечення.

Так як проект розгортається на веб-сторінці, все що нам потрібно для роботи як користувачу, це встановлений браузер, на операційні системи Android/iOS, PC, MAC, LINUX.

Системні вимоги досить низькі, до прикладу нижче приведені мінімальні системні вимоги для Windows такі, як операційна система Windows 10, оперативна пам’ять 1 ГБ, внутрішня пам’ять 100 МБ.

4.4 Інструкція з експлуатації проекту

Для початку роботи з веб-додатком запускається сервер та вводимо в пошуку http://localhost:3000/admin/devices/. З’явиться візуальна частина сайту, які зображені на рисунку 4.1.

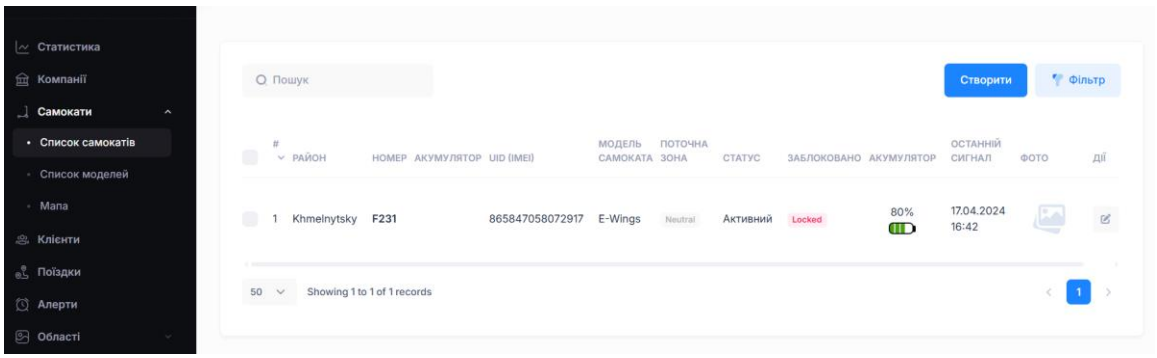


Рисунок 4.1 – Вигляд сторінки

Після того як користувач натисне кнопку «Створити» йому відкриється форма для регістрації самоката у системі вигляд форми на рисунку 4.2.

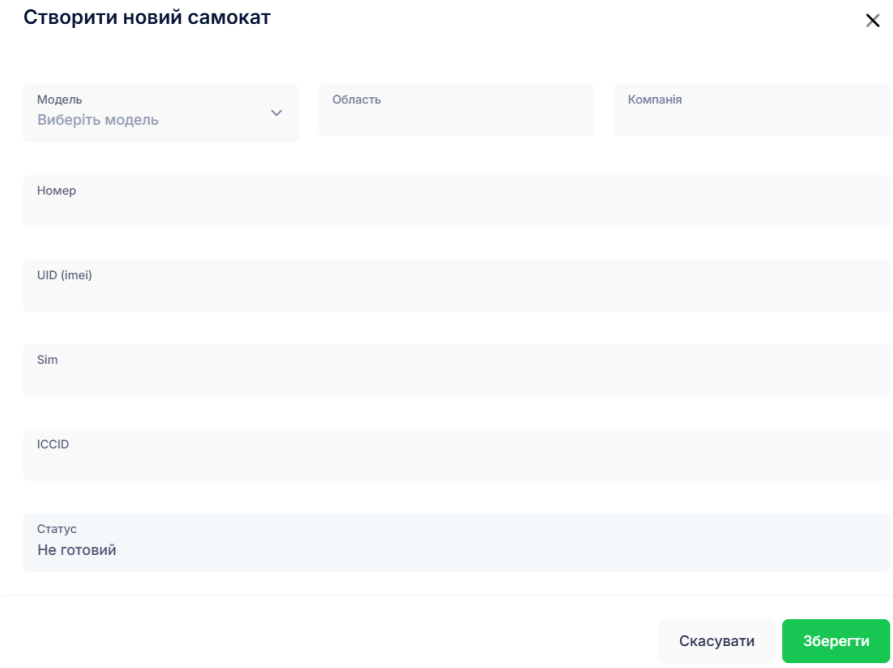


Рисунок 4.2 – Вигляд форми

Якщо натиснути на самий запис в таблиці відкриється детальна інформація про самокат рисунок 4.3, рисунок 4.4, рисонок 4.5.

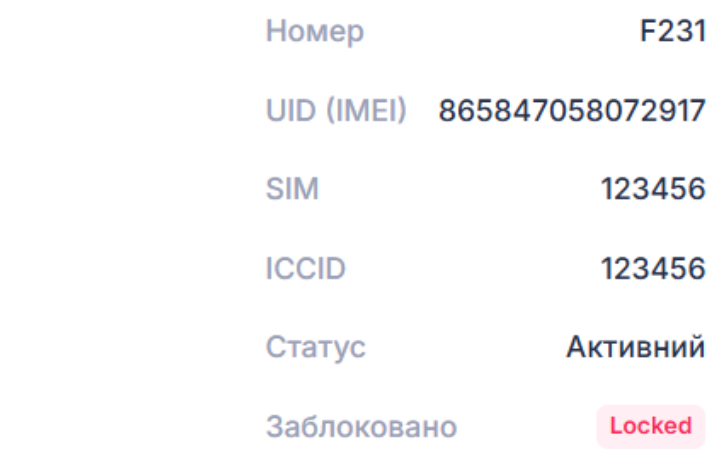


Рисунок 4.3 – Інформація про самокат

Рисунок 4.4 – Додаткова інформація про самокат

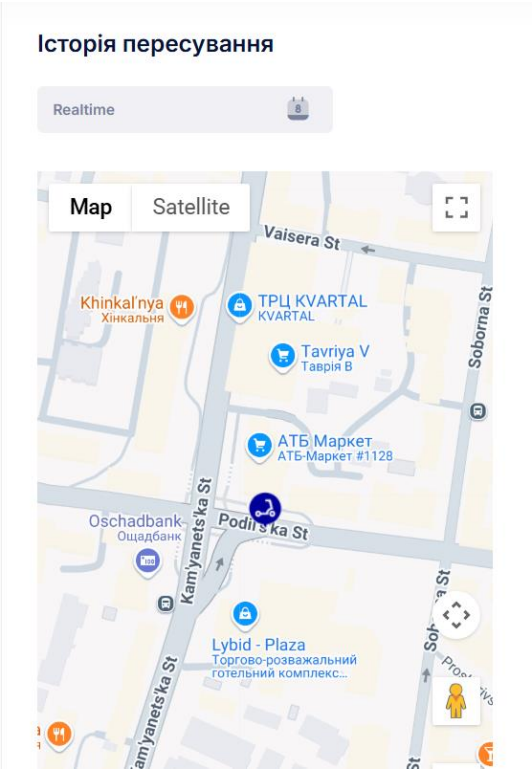


Рисунок 4.5 – Історія переміщення самокат

Коли користувая перейде на сторінку з командами самоката він зможе надіслати самокакату запит на розблокування, заблокування самокату також відкрити кришку батереї, зробити максимальну швидкість таку як йому

завгодно але не більшу ніж може самокат їхати вигляд сторінки показано на рисунку 4.6.

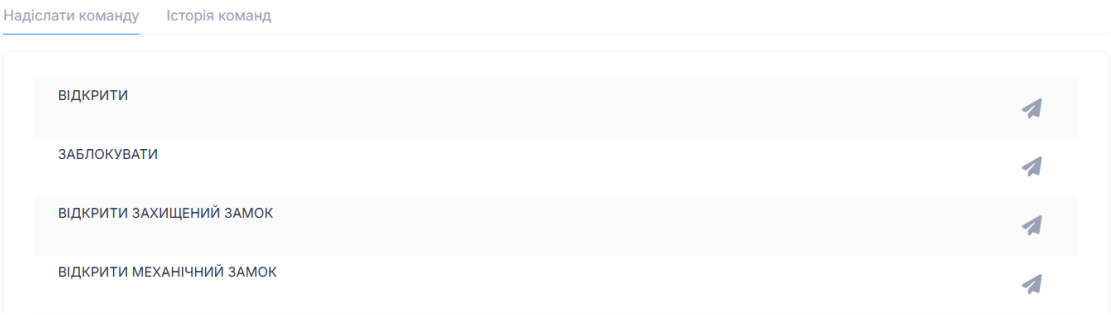


Рисунок 4.6 – Команди самоката

Коли користувач перейде на інформаційну сторінку з мапою то він побачить наступне розташування самокатів на мапі зони в яких самокат може переміщатися, які попередньо були створені та при натискані натискані на маркер йому відкриється інформаційне вікно самоката вигляд мапи на рисунку 4.7.

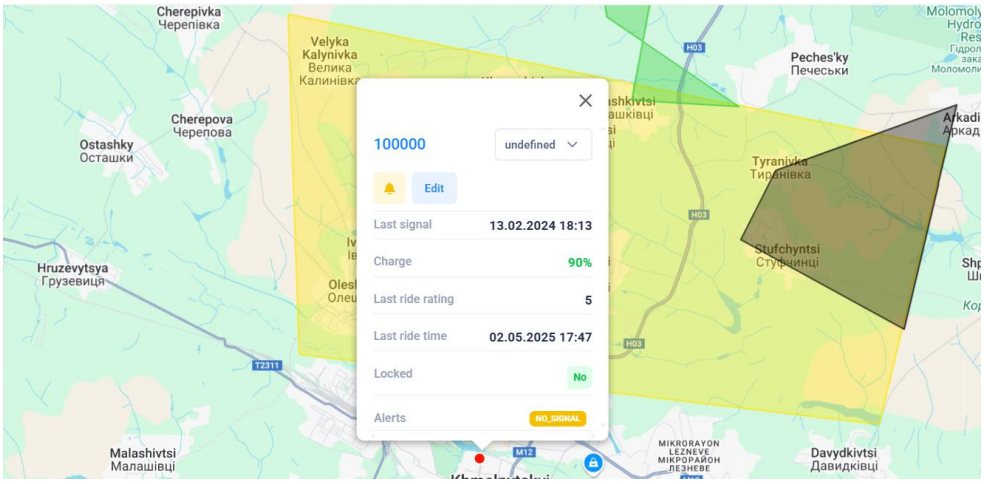


Рисунок 4.7 – Карта

Щоб створити відповідку зону потрібно перейти на вкладку області потім на мапі натискнути праву кнопку миші відкриється інформаційне вікно з типеми зон рисунок 4.8.

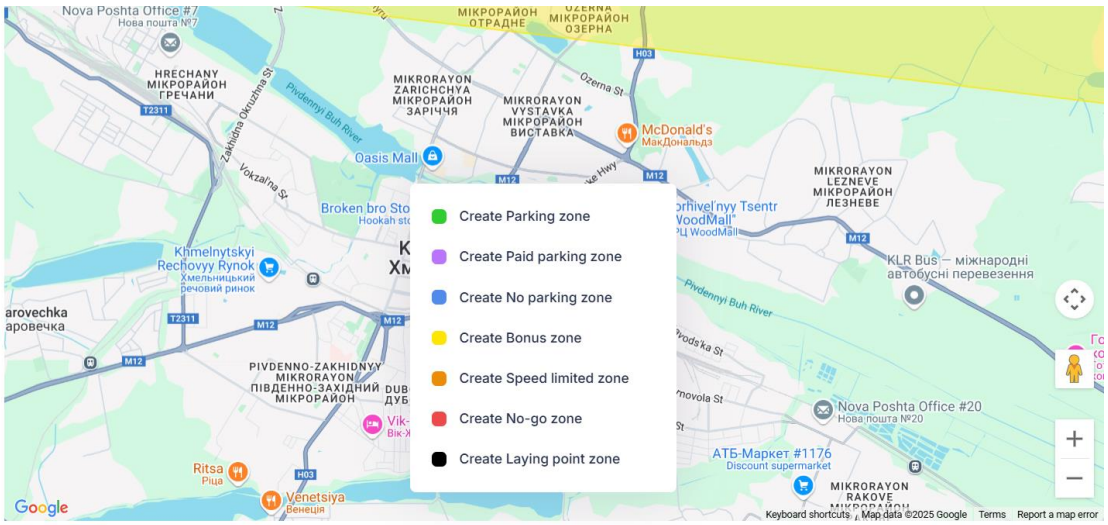


Рисунок 4.8 – Типи зон

ВИСНОВКИ

Розроблена система моніторингу електросамокатів є сучасним інноваційним рішенням, побудованим на принципах клієнт-серверної багаторівневої архітектури з використанням технологій Інтернету речей (IoT). Вона ефективно поєднує апаратну частину (електросамокати з вбудованими контролерами та GPS-модулями) з потужним програмним забезпеченням для візуалізації, аналізу та управління пристроями у реальному часі.

Ключовими особливостями системи є:

- Точне відстеження електросамокатів на карті з використанням API Google Maps, що забезпечує геолокаційний моніторинг та контроль за пересуванням користувачів;
- інтелектуальне зонування: адміністратор має змогу створювати різні типи зон на мапі - дозволене паркування, платне паркування, заборонені для паркування зони, бонусні зони, зони зі зниженою швидкістю, зони із забороною в'їзду тощо. Це дозволяє оптимізувати розміщення та експлуатацію самокатів у місті відповідно до політики керуючої компанії або вимог муніципалітету;
- зрозумілий інтерфейс управління, який дозволяє створювати нові зони через просту інтеракцію на карті - натисканням правої кнопки миші, після чого відкривається меню вибору типу зони. Всі зони мають інтуїтивне кольорове маркування для полегшення навігації та розрізнення;
- можливість аналізу телеметрії за допомогою зібраних даних з пристроїв адміністратори можуть аналізувати маршрути, швидкість, зупинки, місця частого паркування, виявляти зони перевантаження або порушення;
- автоматизований контроль доступу до зон: під час в'їзду до забороненої або платної зони система може автоматично надсилати

сповіщення або обмежувати функціональність самоката.

Завдяки використанню IoT-контролерів (наприклад, на базі ESP8266), система забезпечує збирання та передачу даних у режимі реального часу. Це дозволяє швидко реагувати на позаштатні ситуації, проводити технічну діагностику та автоматизувати обслуговування парку електросамокатів.

У процесі проектування були створені:

- Блок-схеми, що відображають логіку роботи системи та взаємодію між модулями;
- принципова схема IoT-підключення самоката;
- графічний інтерфейс зонування;
- UML-діаграми, які описують структуру та поведінку системи;
- алгоритми визначення зон та прив’язки користувачів до пристроїв.

Отже, створена система повністю відповідає сучасним вимогам до безпеки, зручності та ефективності управління транспортними мікромобільними засобами в умовах міста. Вона має високий потенціал для масштабування, інтеграції з іншими міськими сервісами та подальшого розвитку на базі аналітики даних та штучного інтелекту.

Список використаних джерел

1. AdonisJS Framework Documentation. https://docs.adonisjs.com

2. Node.js Documentation. https://nodejs.org/en/docs/
3. WebSocket - MDN Docs. https://developer.mozilla.org/en-US/docs/Web/API/WebSocket
4. MQTT.org - Lightweight IoT Messaging Protocol. https://mqtt.org
5. ESP8266 Technical Documentation – Espressif. https://www.espressif.com/en/products/socs/esp8266/resources
6. Arduino Documentation (IoT with ESP8266). https://docs.arduino.cc/hardware/nodemcu-esp8266
7. Google Maps JavaScript API Documentation. https://developers.google.com/maps/documentation/javascript/overview
8. Tailwind CSS Documentation. https://tailwindcss.com/docs
9. Formik - React Forms Library. https://formik.org/docs/overview
10. Postman Blog - API Testing for IoT Backends. https://blog.postman.com/tag/iot/
11. OpenStreetMap Documentation. https://wiki.openstreetmap.org/wiki/Main_Page
12. IEEE Internet of Things Journal. https://ieeexplore.ieee.org/xpl/RecentIssue.jsp?punumber=6488907
13. ISO/IEC 30141:2018 - Internet of Things (IoT) Reference Architecture. https://www.iso.org/standard/65695.html
14. W3C Web of Things Architecture. https://www.w3.org/TR/wot-architecture/
15. Minerva, R., Biru, A., & Rotondi, D. (2015). Towards a definition of the Internet of Things (IoT). https://iot.ieee.org/images/files/pdf/IEEE_IoT_Towards_Definition_Internet_of_Things_Revision1_27MAY15.pdf
16. Lee, I., & Lee, K. (2015). The Internet of Things (IoT): Applications, investments, and challenges for enterprises. https://doi.org/10.1016/j.bushor.2015.03.008
17. Vermesan, O., & Friess, P. (Eds.). (2014). Internet of Things: From

Research and Innovation to Market Deployment. https://www.riverpublishers.com/research_details.php?book_id=82

18. McKinsey & Company. (2022). The Internet of Things: Mapping the value beyond the hype. https://www.mckinsey.com/business-functions/mckinsey-digital/our-insights/the-internet-of-things
19. Taneja, M., & Patel, D. (2021). Big Data and IoT for Smart Cities. https://link.springer.com/book/10.1007/978-981-33-4739-2
20. Hassan, Q. F. (2017). Internet of Things: Challenges, Advances, and Applications. https://www.routledge.com/Internet-of-Things-Challenges-Advances-and-Applications/Hassan/p/book/9781498771919
21. Intel Developer Zone - Smart Mobility with IoT. https://www.intel.com/content/www/us/en/internet-of-things/overview.html
22. SmartCitiesWorld - IoT in Urban Mobility. https://www.smartcitiesworld.net/
23. UN Environment Programme - Electric Micro-Mobility. https://www.unep.org/resources/report/electric-two-and-three-wheelers
24. Springer - IoT & Transportation Systems Research. https://link.springer.com/search?query=iot+transportation
25. Transport Reviews Journal - Micromobility Focus. https://www.tandfonline.com/journals/ttrv20
26. World Economic Forum - Urban Mobility Solutions. https://www.weforum.org/agenda/archive/urban-mobility/
27. European Commission - Micro-Mobility Report. https://transport.ec.europa.eu/news-events/news/study-micromobility-european-perspective-2021-12-09_en

Додаток А

```
import DeviceModel from 'Modules/Devices/Models/DeviceModel'
```

```
export default new class DeviceModelClass {

  getSettings(model: DeviceModel) {
    const modelJSON = model?.serialize() ?? {}

    const {
      photo_required,
      charge_low,
      allow_to_start_ride_in_no_go_zone,
      allow_to_end_ride_outside_parking_zone,
      reservation_enabled,
      max_reservation_minutes,
      free_reservation_minutes,
      max_unlocking_distance,
      max_reservation_distance,
      max_allowed_hours_without_rides
    } = modelJSON.settings ?? {}

    return {
      photoRequired: !!photo_required,
      chargeLow: charge_low ? Number(charge_low) : 20,
      allowToStartRideInNoGoZone: !!allow_to_start_ride_in_no_go_zone,
      allowToEndRideOutsideParkingZone: !!allow_to_end_ride_outside_parking_zone,
      reservationEnabled: !!reservation_enabled,
      maxReservationMinutes: max_reservation_minutes ? Number(max_reservation_minutes) : 5,
      freeReservationMinutes: free_reservation_minutes ? Number(free_reservation_minutes) : 0,
      maxUnlockingDistance: max_unlocking_distance ? Number(max_unlocking_distance) : 500,
      maxReservationDistance: max_reservation_distance ? Number(max_reservation_distance) : 3000,
      maxAllowedHoursWithoutRides: max_allowed_hours_without_rides ?
Number(max_allowed_hours_without_rides) : 24
    }
  }
}

import Device from 'Modules/Devices/Models/Device'
import { DeviceAlarm } from 'Modules/Devices/Configs/alarms'



export default new class DeviceClass {
```

```
  checkDeviceAlarms(device: Device): string[] {
    const alarms: DeviceAlarm[] = []
    const { alerts } = device
    if(Array.isArray(alerts) && alerts.length) {

      if(alerts.find((alert) => ['OUT_ZONE_WARN', 'OUT_ZONE'].includes(alert.type))) {
        alarms.push(DeviceAlarm.OUT_ZONE)
      }
      if(alerts.find((alert) => ['ILLEGAL_MOVEMENT'].includes(alert.type))) {
        alarms.push(DeviceAlarm.ILLEGAL_MOVEMENT)
      }
      if(alerts.find((alert) => ['ILLEGAL_REPLACEMENT'].includes(alert.type))) {
        alarms.push(DeviceAlarm.ILLEGAL_REPLACEMENT)
      }
      if(alerts.find((alert) => ['THEFT_ALARM'].includes(alert.type))) {
        alarms.push(DeviceAlarm.THEFT_ALARM)
      }
      if(alerts.find((alert) => ['FALLING'].includes(alert.type))) {
        alarms.push(DeviceAlarm.FALLING)
      }
      if(alerts.find((alert) => ['NO_SIGNAL'].includes(alert.type))) {
        alarms.push(DeviceAlarm.NO_SIGNAL)
      }
    }

    return alarms
  }
}
```

alarms.ts

```
export enum DeviceAlarm {
  LOW_CHARGE = 'low_charge',
  STAY_WITHOUT_RIDE = 'stay_without_ride',
  NO_SIGNAL = 'no_signal',
  OUT_ZONE = 'out_zone',
  ILLEGAL_MOVEMENT = 'illegal_movement',
  ILLEGAL_REPLACEMENT = 'illegal_replacement',
  THEFT_ALARM = 'theft_alarm',
  FALLING = 'falling'
}
```

colors.ts

```
export const colors = [
```

```
            '#008FFB',
            '#00E396',
            '#FEB019',
            '#FF4560',
            '#775DD0',
            '#3F51B5',
            '#03A9F4',
            '#4CAF50',
            '#F9CE1D',
            '#FF9800',
            '#33B2DF',
            '#546E7A',
            '#D4526E',
            '#13D8AA',
            '#A5978B'
          ]
```

commands.ts

```
export default [
  'UNLOCK',
  'LOCK',
  'UNLOCK_ENCHANTED',
  'UNLOCK_MECHANICAL_LOCK',
  'LOCK_MECHANICAL_LOCK',
  'REBOOT',
  'POWER_OFF',
  'UPDATE_GEO_INFO',
  'WARN',
  'HEADLIGHT_ON',
  'HEADLIGHT_OFF',
  'REARLIGHT_ON',
  'REARLIGHT_ON_FORCED',
  'REARLIGHT_OFF',
  'ENGINE_ON',
  'SET_VEHICLE_IN_NORMAL_MODE',
  'SET_VEHICLE_IN_TEST_MODE',
  'SPEED_LIMIT',
  'GASDISSABLE',
  'GASENABLE',
  'UNLOCKLEAD'
]
```

statuses.ts

```
export enum DeviceStatus {
  AVAILABLE = 'available',
  RESERVED = 'reserved',
  IN_USE = 'in_use',
  PAUSE = 'pause',
  DISCHARGED = 'discharged',
  NO_GSM = 'no_gsm',
  NO_GPS = 'no_gps',
  NEED_INVESTIGATION = 'need_investigation',
  TRANSPORTATION = 'transportation',
  MAINTENANCE = 'maintenance',
  STOLEN = 'stolen',
  STORAGE = 'storage',
```

```
  READY = 'ready',
  NOT_READY = 'not_ready',
  LAUNCH_PROCESSING = 'launch_processing',
  SPARE_IOT = 'spare_iot',
  BROKEN_IOT = 'broken_iot',
  DEAD_IOT = 'dead_iot',
  STAY_WITHOUT_RIDE = 'stay_without_ride'
}

export default [
  {
    value: DeviceStatus.AVAILABLE,
    color: '#6AA84F'
  },
  {
    value: DeviceStatus.RESERVED,
    color: '#00FFFF'
  },
  {
    value: DeviceStatus.LAUNCH_PROCESSING,
    color: '#00FFFF'
  },
  {
    value: DeviceStatus.IN_USE,
    color: '#00FFFF'
  },
  {
    value: DeviceStatus.PAUSE,
    color: '#00FFFF'
  },
  {
    value: DeviceStatus.DISCHARGED,
    color: '#000000'
  },
  {
    value: DeviceStatus.NO_GSM,
    color: '#FF00FF'
  },
  {
    value: DeviceStatus.NO_GPS,
    color: '#EAD1DC'
  },
  {
    value: DeviceStatus.NEED_INVESTIGATION,
    color: '#FF0000'
  },
  {
    value: DeviceStatus.TRANSPORTATION,
    color: '#A64D79'
  },
  {
    value: DeviceStatus.MAINTENANCE,
    color: '#3C78D8'
  },
  {
    value: DeviceStatus.STOLEN,
    color: '#000000'
  },
  {
    value: DeviceStatus.STORAGE,
    color: '#000000'   },
  {
```

```
    value: DeviceStatus.READY,
    color: '#FFFF00'
  },
  {
    value: DeviceStatus.NOT_READY,
    color: '#000000'
  },
  {
    value: DeviceStatus.SPARE_IOT,
    color: '#000000'
  },
  {
    value: DeviceStatus.BROKEN_IOT,
    color: '#000000'
  },
  {
    value: DeviceStatus.DEAD_IOT,
    color: '#000000'
  },
  {
    value: DeviceStatus.STAY_WITHOUT_RIDE,
    color: '#000000'   },
]
```

DeviceActivityController.ts

```
import Device from 'Modules/Devices/Models/Device'
import Env from '@ioc:Adonis/Core/Env'

const axios = require('axios')

export default class DeviceActivityController {

  async list({ request }) {
    const { id, from, to } = request.all()

    const device = await Device.findOrFail(id)

    const res = await Device.findOrFail(id)

    const { success, list } = res.data

    if(success) {
      return {
        data: list.filter((item) => item.command).map((item) => {
          const { userId, userEmail } = item.data ?? {}

          return {
            date: item.date,
            customer: userEmail ? userEmail : userId ? `user_id: ${userId}` : 'SYSTEM',
            action: item.command
          }
        })
      }
    }

    return {
      data: []
    }
  }
}
```

DeviceBatteryHistoryController.ts

```
'use strict'
```

```
import { DataTable } from '@ioc:Adm/DataTable'
import Database from '@ioc:Adonis/Lucid/Database'

export default class DeviceBatteryHistoryController {

  async list({ request, params }) {
    const { id } = params

    const query = Database
      .from('device_battery_histories')
      .select(
        'device_battery_histories.id',
        'device_battery_histories.battery_id',
        'batteries.number as battery_number',
        'device_battery_histories.device_id',
        'device_battery_histories.installation_time',
        'device_battery_histories.removal_time',
      )
      .leftJoin('batteries', 'device_battery_histories.battery_id', 'batteries.id')
device_battery_histories.installed_by_employee_id = u1.id') device_battery_histories.removed_by_employee_id =
u2.id')
          .where('device_battery_histories.device_id', id)

    const table = new DataTable(query, request)

    return await table.make()
  }
}
DeviceModelsController.ts
import Route from '@ioc:Adonis/Core/Route'
import { DataTable, TableBuilder } from '@ioc:Adm/DataTable'

import Database from '@ioc:Adonis/Lucid/Database'

import DeviceModel from 'Modules/Devices/Models/DeviceModel'
import Area from 'Modules/Areas/Models/Area'
import Notify from '@ioc:Adm/Notify'
import Config from '@ioc:Adonis/Core/Config'
import { rules, schema } from '@ioc:Adonis/Core/Validator'
import Device from 'Modules/Devices/Models/Device'
import { DateTime } from 'luxon'

import Event from '@ioc:Adonis/Core/Event'
import File from 'App/Classes/File'
import ManagerDeviceModel from 'Modules/Managers/Models/ManagerDeviceModel'

export default class DeviceModelsController {

  async page({ view, acl, i18n }) {

    const table = new TableBuilder('deviceModels')

    if (acl.hasPermissions('device_models.create')) {
      table.setButtons([
        `<a href="${Route.makeUrl('Devices.models.edit')}" class="btn btn-primary no-
history">${i18n.formatMessage('admin.general.create')}</a>`
      ])
    }

    table.setColumns([
      { title: '#', width: '5%' },
      { title: i18n.formatMessage('admin.device_models.name'), width: '' },
```

```
      { title: i18n.formatMessage('admin.areas.area'), width: '' },
      { title: i18n.formatMessage('admin.general.actions'), width: '1%' }
    ])

    return view.render('Devices::models/index', {
      pageTitle: i18n.formatMessage('admin.devices.page'),
      breadcrumbs: [
        { name: '<i class="ki-duotone ki-home fs-3 text-gray-400"></i>', url: Route.makeUrl('admin.dashboard') },
        { name: i18n.formatMessage('admin.device_models.list') }
      ],
      table: await table.build()
    })
  }

  async list({ request, auth, response }) {
    const { user: manager } = await auth.use('admin')

    const query = Database
      .from('device_models')
      .select('device_models.id',
        'device_models.name as model_name',
        'areas.name as area_name',
      )
      .leftJoin('areas', 'device_models.area_id', 'areas.id')
      .whereIn('device_models.company_id', await manager.getEnabledCompanyIds())
      .whereIn('device_models.id', await manager.getDeviceModelIds())

    const table = new DataTable(query, request)
    const res = await table.make()

    return response.json(res)
  }

  async edit({ params, auth, i18n, view }) {
    const { id } = params

    let deviceModel: DeviceModel | null
    if (id) {
      deviceModel = await DeviceModel.find(id)

      if (!deviceModel) {
        return Notify.error('Model not found', {})
      }
    } else {
      deviceModel = new DeviceModel()
    }

    const { user: manager } = await auth.use('admin')

    const areas = await Area
      .query()
      .select('areas.*')
      .leftJoin('company_manager', 'areas.company_id', 'company_manager.company_id')
      .where('company_manager.manager_id', manager.id)
      .preload('company')

    const providers = Config.get('providers')

    return {
      modal: {
        title: id ? i18n.formatMessage('admin.device_models.edit_model') :
  i18n.formatMessage('admin.device_models.create_new_model'),
```

```
      content: await view.render('Devices::models/form', {
        deviceModel,
        areas,
        providers
      }),
      cancel: i18n.formatMessage('admin.general.cancel'),
      submit: i18n.formatMessage('admin.general.save'),
      width: '800px'
    },
    success: true
  }
}

async save({ request, auth }) {
  if (!!request.all().tariffs?.has_dynamic_prices) {
    request.all().has_dynamic_prices = true
  }

  const saveDeviceModelSchema = schema.create({
    id: schema.number.optional(),
    area_id: schema.number(),
    name: schema.string({}, [
      rules.minLength(3),
      rules.maxLength(255)
    ]),
    provider: schema.string(),
    tariffs: schema.object().members({
      price: schema.number(),
      price_unlock: schema.number(),
      price_pause: schema.number(),
      price_reservation: schema.number(),

      has_special_price: schema.boolean.optional(),
      idle_hours_for_special_price: schema.number(),
      special_price_discount: schema.number(),
      out_zone_fine: schema.number([
        rules.range(0, 1000)
      ]),
      has_dynamic_prices: schema.boolean.optional()
    }),
    settings: schema.object().members({
      charge_low: schema.number(),
      charge_medium: schema.number(),
      allowed_charge_for_rent: schema.number(),
      default_speed: schema.number(),
      denied_area_time_limit: schema.number(),
      denied_area_warning_duration: schema.number(),

      finish_ride_if_not_moving: schema.boolean.optional(),
      photo_required: schema.boolean.optional(),
      allow_to_end_ride_outside_parking_zone: schema.boolean.optional(),
      allow_to_start_ride_in_no_go_zone: schema.boolean.optional(),
      reservation_enabled: schema.boolean.optional(),
      max_unlocking_distance: schema.number([
        rules.range(0, 10000000)
      ]),
      max_reservation_distance: schema.number([
        rules.range(0, 100000)
      ]),
      max_reservation_minutes: schema.number([
        rules.range(0, 1440)
      ]),
```

```
      free_reservation_minutes: schema.number([
        rules.range(0, 1440)
      ]),
      max_allowed_hours_without_rides: schema.number([
        rules.range(1, 720)
      ]),
      hide_in_no_gps_status: schema.boolean.optional(),
      noise_activated: schema.boolean.optional(),
      noise_displacement: schema.number(),
      noise_repetition: schema.number(),
      noise_alert_at: schema.number(),
      days: schema.number.optional(),
      kilometers: schema.number.optional(),
      rides: schema.number.optional(),
      noise_frame: schema.number(),
      no_go_speed_limit: schema.number(),
      no_go_area_warning_interval: schema.number(),
      allow_to_end_ride_at_no_parking_zone: schema.boolean.optional(),
      allow_to_rent_device_at_no_gps_status: schema.boolean.optional()
    }),
    dynamic_prices:
schema.array.optional([rules.requiredIfExists('has_dynamic_prices')]).members(schema.object().members({
      weekday: schema.string(),
      time: schema.string(),
      price: schema.number(),
      price_unlock: schema.number(),
      price_pause: schema.number()
    })),
    replaceable_battery: schema.boolean.optional(),
  })

  const {
    id,
    area_id,
    name,
    provider,
    tariffs,
    settings,
    dynamic_prices,
    replaceable_battery
  } = await request.validate({ schema: saveDeviceModelSchema })

  let deviceModel: DeviceModel | null
  let devices
  if (!id) {
    deviceModel = new DeviceModel()

    const area = await Area.find(area_id)
    if (!area) {
      return Notify.error('Територію не знайдено')
    }

    deviceModel.areaId = area.id
    deviceModel.companyId = area.companyId
  }
  else {
    deviceModel = await DeviceModel.find(id)
    devices = await Device.query().where('device_model_id', id).exec()
    if (!deviceModel) {
      return Notify.error('Model not found', {})
    }
  }
```

```
deviceModel.name = name
deviceModel.provider = provider
deviceModel.replaceableBattery = !!replaceable_battery
deviceModel.tariffs = {
  price: Number(tariffs.price),
  price_time: 60, // todo
  price_unlock: Number(tariffs.price_unlock),
  price_pause: Number(tariffs.price_pause),
  price_reservation: Number(tariffs.price_reservation),
  has_special_price: !!tariffs.has_special_price,
  idle_hours_for_special_price: Number(tariffs.idle_hours_for_special_price),
  special_price_discount: Number(tariffs.special_price_discount),
  out_zone_fine: Number(tariffs.out_zone_fine),
  has_dynamic_prices: !!tariffs.has_dynamic_prices
}
deviceModel.settings = {
  charge_low: Number(settings.charge_low),
  charge_medium: Number(settings.charge_medium),
  allowed_charge_for_rent: Number(settings.allowed_charge_for_rent),
  default_speed: Number(settings.default_speed),
  denied_area_time_limit: Number(settings.denied_area_time_limit),
  denied_area_warning_duration: Number(settings.denied_area_warning_duration),
  finish_ride_if_not_moving: !!settings.finish_ride_if_not_moving,
  photo_required: !!settings.photo_required,
  allow_to_end_ride_outside_parking_zone: !!settings.allow_to_end_ride_outside_parking_zone,
  allow_to_start_ride_in_no_go_zone: !!settings.allow_to_start_ride_in_no_go_zone,
  reservation_enabled: !!settings.reservation_enabled,
  max_unlocking_distance: Number(settings.max_unlocking_distance),
  max_reservation_distance: Number(settings.max_reservation_distance),
  max_reservation_minutes: Number(settings.max_reservation_minutes),
  free_reservation_minutes: Number(settings.free_reservation_minutes),
  max_allowed_hours_without_rides: Number(settings.max_allowed_hours_without_rides),
  hide_in_no_gps_status: !!settings.hide_in_no_gps_status,
  noise_activated: !!settings.noise_activated,
  noise_displacement: Number(settings.noise_displacement),
  noise_repetition: Number(settings.noise_repetition),
  noise_alert_at: Number(settings.noise_alert_at),
  days: Number(settings.days),
  kilometers: Number(settings.kilometers),
  rides: Number(settings.rides),
  noise_frame: Number(settings.noise_frame),
  no_go_speed_limit: Number(settings.no_go_speed_limit),
  no_go_area_warning_interval: Number(settings.no_go_area_warning_interval),
  allow_to_end_ride_at_no_parking_zone: !!settings.allow_to_end_ride_at_no_parking_zone,
  allow_to_rent_device_at_no_gps_status: !!settings.allow_to_rent_device_at_no_gps_status
}

if (deviceModel.tariffs.has_dynamic_prices) {
  const dynamicPrices = dynamic_prices.map((dynamicPrice) => {
    const { weekday, price, price_unlock, price_pause } = dynamicPrice
    const [from_time, to_time] = dynamicPrice.time.split(' - ')

    return {
      weekday,
      from_time,
      to_time,
      price,
      price_unlock,
      price_pause
    }
  })
```

```
    deviceModel.dynamicPrices = JSON.stringify(dynamicPrices)
  }

  if (deviceModel.settings.days > 0 || deviceModel.settings.kilometers > 0 || deviceModel.settings.rides > 0) {
    for (let device of devices) {
      if (!device.$original.servicedAt || device.$original.servicedAt == null) {
        device.servicedAt = DateTime.now();
        device.needsMaintenance = false;
        await device.save();
      }
    }
  }

  const photo = request.file('photo')
  if (photo) {
    const { success, filePath } = await File.saveImage(photo, 'device_models')
    if (!success) {
      return Notify.error('Помилка при завантаженні зображення', {})
    }

    deviceModel.photo = filePath
  }

  await deviceModel.save()

  if (!id) {
    const { user: manager } = await auth.use('admin')

    await ManagerDeviceModel.create({
      deviceModelId: deviceModel.id,
      managerId: manager.id
    })
  }

  Event.emit('updateBrokerModels', {})

  return Notify.success('Saved', {})
}

async delete({ params }) {
  const { id } = params
  const deviceModel = await DeviceModel.find(id)

  if (!deviceModel) {
    return Notify.error('Something went wrong. Model not found', {})
  }

  await deviceModel.delete()

  return Notify.success('Deleted', {})
}

async dynamicPricesList({ request }) {
  const { device_model_id } = request.all()

  const deviceModel = await DeviceModel.find(device_model_id)

  if (!deviceModel) {
    return Notify.error('Something went wrong. Model not found', {})
  }
```

```
    const dynamicPrices = deviceModel.dynamicPrices || []

    dynamicPrices.forEach((dynamicPrice) => {
      const { from_time, to_time } = dynamicPrice
      dynamicPrice.time = `${from_time} - ${to_time}`
    })

    return {
      success: true,
      list: dynamicPrices
    }
  }
}
```

DeviceMovementHistoryController.ts

```
import Device from 'Modules/Devices/Models/Device'
import Area from 'Modules/Areas/Models/Area'
import Setting from 'Modules/Settings/Models/Setting'
import Config from '@ioc:Adonis/Core/Config'
import Notify from '@ioc:Adm/Notify'
import Route from '@ioc:Adonis/Core/Route'
import Database from '@ioc:Adonis/Lucid/Database'

export default class DeviceMovementHistoryController {

  async page({ request, params, view, i18n }) {
    const { id } = params
    const { alert_id: alertId } = request.all()

    const device = await Device.find(id)
    if (!device) {
      return Notify.error('Скутер не знайдено', {})
    }

    const { lat, lng } = device
    if(!lat || !lng) {
      return Notify.error('Координати не знайдено', {})
    }

    const area = await Area.query().where('id', device.areaId).preload('zones').preload('city').first()
    if (!area) {
      return Notify.error('Територія не знайдена', {})
    }

    const zonesList = Config.get('zones')
    const zoneColorsSettings = await Setting.findBy('name', 'zone_colors')
    const zoneColors = zoneColorsSettings?.value ? JSON.parse(zoneColorsSettings.value) : {}
    zonesList.forEach((zone) => {
      zone.color = zoneColors?.[zone.type] ?? zone.color
    })

    return view.render('Devices::devices/movement-history/index', {
      pageTitle: i18n.formatMessage('admin.devices.devices'),
      breadcrumbs: [
        { name: '<i class="ki-duotone ki-home fs-3 text-gray-400"></i>', url: Route.makeUrl('admin.dashboard') },
        { name: i18n.formatMessage('admin.devices.list'), url: Route.makeUrl('Devices.devices.page') },
        { name: `Scooter ${device.number}`, url: Route.makeUrl('Devices.devices.show', { id: device.id }) },
        { name: i18n.formatMessage('admin.devices.movementHistory') }
      ],
      area,
      zonesList,
      device,
      alertId: alertId || null
```

```
    })
  }


  async list({ params, request }) {
    const { id } = params
    const { from, to } = request.all()

    const device = await Device.find(id)
    if (!device) {
      return Notify.error('Vehicle not found', {})
    }

    let history: any = await Database
      .from('devices_histories')
      .select('id', 'lat', 'lng', 'charge', 'locked', 'date', 'noise')
      .where('uid', device.uid)
      .whereBetween('date', [from, to])
      .whereNotNull('lat')
      .whereNotNull('lng')
      .orderBy('id', 'asc')

    // const history = Config.get('history')

    return {
      success: true,
      history
    }
  }

  async getCurrentGeo({ params }) {
    const { id } = params

    const device = await Device.find(id)
    if (!device) {
      return Notify.error('Vehicle not found', {})
    }

    const { lat, lng, activeAt } = device
    if(!lat || !lng) {
      return Notify.error('Coordinates not found', {})
    }

    return {
      success: true,
      lat,
      lng,
      date: activeAt
    }
  }
}
DeviceNotesController.ts
import { DataTable } from '@ioc:Adm/DataTable'
import Database from '@ioc:Adonis/Lucid/Database'
import Notify from '@ioc:Adm/Notify'
import DeviceNote from 'Modules/Devices/Models/DeviceNote'

export default class DeviceNotesController {

  async list({ request, response, params }) {
    const { id } = params
```

```
    const query = Database
      .from('device_notes')
      .select(
        'device_notes.id',
        'device_notes.text',
        'device_notes.admin_id',
        'device_notes.created_at',
        'managers.name as admin_name',
        Database.raw("string_agg(distinct roles.name, ',') as roles_names"),
      )
      .join('managers', 'device_notes.admin_id', 'managers.id')
      .leftJoin('role_manager', 'role_manager.manager_id', 'managers.id')
      .leftJoin('roles', 'roles.id', 'role_manager.role_id')
      .where('device_notes.device_id', id)
      .groupBy(
        'device_notes.id',
        'device_notes.text',
        'device_notes.admin_id',
        'device_notes.created_at',
        'managers.name',
      )

    const table = new DataTable(query, request)
    const res = await table.make()

    return response.json(res)
  }

  async save({ request, auth }) {
    const input = request.all()
    const { user: manager } = await auth.use('admin')

    await DeviceNote.create({
      deviceId: input.id,
      text: input.text,
      adminId: manager.id
    })

    return Notify.success('Saved')
  }
}
```

DeviceRidesController.ts

```
const moment = require('moment/moment')
import { DataTable } from '@ioc:Adm/DataTable'
import Database from '@ioc:Adonis/Lucid/Database'
import Ride from 'Modules/Rides/Models/Ride'
import RideClass from 'App/Classes/RideClass'

export default class DeviceRidesController {

  async list({ request, response, params }) {
    const { id } = params

    const query = Database
      .from('rides')
      .select(
        'rides.id',
        'rides.type',
        'rides.user_id',
        'rides.cost',
        'rides.status',
```

```
        'rides.time_start',
        'rides.time_end',
        'rides.distance',
        'rides.duration',
        'devices.id as device_id',
        'devices.number as device_number',
        'users.id as user_id',
        'users.name as user_name',
        'ride_feedback.stars',
        'ride_feedback.comment'
      )
      .leftJoin('users', 'users.id', 'rides.user_id')
      .leftJoin('devices', 'devices.id', 'rides.device_id')
      .leftJoin('ride_feedback', 'rides.id', 'ride_feedback.ride_id')
      // .where('rides.type', Ride.typeRide)
      .where('rides.device_id', id)

    const table = new DataTable(query, request)
    const res: any = await table.make()

    for (const ride of res.data) {
      if(ride.type === 'ride') {
        if(ride.status !== Ride.statusFinished) {
          ride.duration = moment.utc(RideClass.getRideDuration(ride, 'milliseconds')).format('HH:mm:ss')
        } else {
          ride.duration = moment.utc(ride.duration * 1000).format('HH:mm:ss')
        }
      }
      else {
        ride.duration = ''
      }
    }

    return response.json(res)
  }
}
```

DevicesController.ts

```
\import Config from '@ioc:Adonis/Core/Config'
import Route from '@ioc:Adonis/Core/Route'

import { DataTable, TableBuilder } from '@ioc:Adm/DataTable'
import Database from '@ioc:Adonis/Lucid/Database'

import Device from 'Modules/Devices/Models/Device'
import Notify from '@ioc:Adm/Notify'
import DeviceCommunicator from 'Modules/Devices/Services/DeviceCommunicator'
import Area from 'Modules/Areas/Models/Area'
import DeviceModel from 'Modules/Devices/Models/DeviceModel'
import Setting from 'Modules/Settings/Models/Setting'
const zonesList = Config.get('zones')
import AreaZoneClass from 'Modules/Areas/Classes/AreaZoneClass'
import City from 'Modules/Areas/Models/City'
import { schema, rules } from '@ioc:Adonis/Core/Validator'
import DeviceStatusChangeLog from 'Modules/Logs/Models/DeviceStatusChangeLog'
import Event from '@ioc:Adonis/Core/Event'
import DeviceClass from 'Modules/Devices/Classes/DeviceClass'
import { DeviceStatus } from 'Modules/Devices/Configs/statuses'
import { constantCase } from 'change-case'
import Ride, { RideStatus } from 'Modules/Rides/Models/Ride'
import RideClass from 'App/Classes/RideClass'
import OrderClass from 'App/Classes/OrderClass'
```

```
export default class DevicesController {
  ignoringStatuses = [DeviceStatus.IN_USE, DeviceStatus.RESERVED, DeviceStatus.PAUSE,
DeviceStatus.LAUNCH_PROCESSING]

  async page({ view, i18n, auth, acl }) {
    const table = new TableBuilder('devices')

    table.setColumns([
      { title: '', width: '2%' },
      { title: '#', width: '5%' },
      { title: i18n.formatMessage('admin.devices.area'), width: '' },
      { title: i18n.formatMessage('admin.devices.number'), width: '' },
      { title: i18n.formatMessage('admin.batteries.battery'), width: '' },
      { title: i18n.formatMessage('admin.devices.uid'), width: '' },
      { title: i18n.formatMessage('admin.device_models.device_model'), width: '' },
      { title: i18n.formatMessage('admin.devices.current_zone'), width: '' },
      { title: i18n.formatMessage('admin.devices.status'), width: '' },
      { title: i18n.formatMessage('admin.devices.locked'), width: '' },
      { title: i18n.formatMessage('admin.devices.charge'), width: '' },
      { title: i18n.formatMessage('admin.devices.last_signal'), width: '' },
      { title: i18n.formatMessage('admin.rides.photo'), width: '' },
      { title: i18n.formatMessage('admin.general.actions'), width: '1%' }
    ])

    table.setActions(await view.render('Devices::devices/_components/actions'))

    const { user: manager } = await auth.use('admin')

    const deviceModels = await DeviceModel
      .query()
      .select('device_models.*')
      .whereIn('device_models.company_id', await manager.getEnabledCompanyIds())
      .preload('company')

    const deviceStatuses: any = Device.getAllStatuses().filter((s) =>
![DeviceStatus.LAUNCH_PROCESSING].includes(s.value))

    table.setFilter(await view.render('Devices::devices/_components/filterForm', {
      deviceStatuses,
      deviceModels
    }))

    if (acl.hasPermissions('devices.create')) {
      table.setButtons([`<a href="${Route.makeUrl('Devices.devices.edit')}" class="btn btn-primary no-
history">${i18n.formatMessage('admin.general.create')}</a>`])
    }

    return view.render('Devices::devices/page', {
      pageTitle: i18n.formatMessage('admin.devices.page'),
      breadcrumbs: [
        { name: '<i class="ki-duotone ki-home fs-3 text-gray-400"></i>', url: Route.makeUrl('admin.dashboard') },
        { name: i18n.formatMessage('admin.devices.list') }
      ],
      table: await table.build()
    })
  }

  async map({ auth, view, i18n }) {
    const { user: manager } = await auth.use('admin')

    const areas = await Area
      .query()
```

```
      .select('areas.*')
      .whereIn('areas.company_id', await manager.getEnabledCompanyIds())
      .preload('city')
      .preload('zones')

    const cities: City[] = []
    areas.forEach((area) => {
      if (!cities.find((city) => city.id === area.cityId)) {
        cities.push(area.city)
      }
    })

    const deviceModels = await DeviceModel
      .query()
      .select('device_models.*')
      // .whereIn('device_models.company_id', await manager.getEnabledCompanyIds())
      .whereIn('device_models.id', await manager.getDeviceModelIds())
      .preload('company')

    const zoneColorsSettings = await Setting.findBy('name', 'zone_colors')
    const zoneColors = zoneColorsSettings?.value ? JSON.parse(zoneColorsSettings.value) : {}
    zonesList.forEach((zone) => {
      zone.color = zoneColors?.[zone.type] ?? zone.color
    })

    const deviceStatuses: any = Device.getAllStatuses().filter((s) =>
![DeviceStatus.LAUNCH_PROCESSING].includes(s.value))

    return view.render('Devices::devices/map', {
      pageTitle: i18n.formatMessage('admin.devices.page'),
      breadcrumbs: [
        { name: '<i class="ki-duotone ki-home fs-3 text-gray-400"></i>', url: Route.makeUrl('admin.dashboard') },
        { name: i18n.formatMessage('admin.devices.map') }
      ],
      areas,
      cities,
      deviceModels,
      deviceStatuses,
      zonesList
    })
  }

  devicesListQueryBuilder(query, { request, companyIds, deviceModelIds, filter = {} }: any) {
    const { city_id } = request.all()
    const { statuses, model_ids, charge, last_ride_hours, models, last_mot, no_battery } = filter

    query
      .select(
        'devices.id',
        'areas.name as area_name',
        'devices.uid',
        'devices.area_id',
        'devices.device_model_id',
        'device_models.name as model_name',
        'devices.number',
        'devices.locked',
        'devices.lat',
        'devices.lng',
        'devices.status',
        'devices.active_at',
        'devices.charge',
        'devices.real_charge',
```

```
      'devices.available_from',
      'devices.last_ride_photo',
      'devices.needs_maintenance'
    )
    .leftJoin('device_models', 'devices.device_model_id', 'device_models.id')
    .leftJoin('areas', 'devices.area_id', 'areas.id')
    .whereIn('devices.company_id', companyIds)
    .whereIn('devices.device_model_id', deviceModelIds)
    .whereNull('devices.deleted_at')
  // .groupBy(['devices.id', 'device_models.name', 'areas.name'])

  if (city_id) {
    query.where('areas.city_id', city_id)
  }
  if (Array.isArray(statuses) && statuses.length) {
    query.whereIn('devices.status', statuses)
  }
  if (Array.isArray(models) && models.length) {
    query.whereIn('devices.device_model_id', models)
  }
  if (Array.isArray(model_ids) && model_ids.length) {
    query.whereIn('devices.device_model_id', model_ids)
  }
  if (last_ride_hours) {
    const { from, to } = last_ride_hours

    if (from) {
      query.whereRaw(`devices.available_from <= now() - interval '${from} hours'`)
    }
    if (to) {
      query.whereRaw(`devices.available_from >= now() - interval '${to} hours'`)
    }
  }
  if (charge) {
    const { from, to } = charge
    if (from) {
      query.where('devices.charge', '>=', from)
    }
    if (to) {
      query.where('devices.charge', '<=', to)
    }
  }
  if (last_mot) {
    const { from, to } = last_mot
    if (from) {
      query.whereRaw(`devices.serviced_at >= now() - interval '${from} days'`)
    }
    if (to) {
      query.whereRaw(`devices.serviced_at >= now() - interval '${to} days'`)
    }
  }
  if (no_battery) {
    query.whereNull('devices.battery_id')
  }

  return query
}

async listForDatatable({ request, auth, i18n, view, response }) {
  const filter = request.input('filter') ?? {}

  const { user: manager } = await auth.use('admin')
```

```
    const companyIds = await manager.getEnabledCompanyIds()
    const deviceModelIds = await manager.getDeviceModelIds()

    const query = Database
      .connection('pgReader')
      .from('devices')
      .leftJoin('batteries', 'devices.battery_id', 'batteries.id')
      .select('devices.*', 'batteries.number as number_battery')

    this.devicesListQueryBuilder(query, { request, companyIds, deviceModelIds, filter })

    const table = new DataTable(query, request)
    const res: any = await table.make()

    for (const element of res.data) {
      element.charge = await view.render('Devices::devices/_components/batteryRange', {
        device: element,
        width: '70px'
      })
      element.status = i18n.formatMessage(`admin.devices.statuses.${element.status}`)
      element.current_zone = AreaZoneClass.getCurrentZone(element)
    }

    return response.json(res)
  }

  async listForMap({ request, auth, i18n }) {
    const { minLat, minLng, maxLat, maxLng } = request.all()

    const filter = request.input('filter') ?? {}
    delete filter.statuses
    delete filter.models

    const searchValue = request.input('searchValue') ?? ''

    const { user: manager } = await auth.use('admin')
    const companyIds = await manager.getEnabledCompanyIds()
    const deviceModelIds = await manager.getDeviceModelIds()

    const query = Device
      .query()
      .whereBetween('lat', [minLat, maxLat])
      .andWhereBetween('lng', [minLng, maxLng])
      .preload('model')
      .preload('alerts', (builder) => {
        builder.where('closed', false)
        builder.whereIn('type', ['OUT_ZONE', 'ILLEGAL_MOVEMENT', 'ILLEGAL_REPLACEMENT',
'THEFT_ALARM', 'FALLING', 'LOW_CHARGE'])
      })

    this.devicesListQueryBuilder(query, { request, companyIds, deviceModelIds, filter })

    if (searchValue) {
      if (searchValue.split(' ').length > 1) {
        query.whereIn('devices.number', searchValue.split(' '))
      } else {
        query.where(function (builder) {
          builder.whereRaw(`devices.number like '${searchValue}%'`)
          builder.orWhereRaw(`devices.uid like '%${searchValue}%'`)
        })
      }
    }
```

```
    const devices: Device[] = await query
    devices.forEach((device) => {
      device.alarms = DeviceClass.checkDeviceAlarms(device).map((type) =>
i18n.formatMessage(`admin.devices.alarms.${constantCase(type)}`))
    })

    return {
      success: true,
      data: devices
    }
  }

  async show({ params, i18n, view }) {
    const { id } = params

    const device = await Device
      .query()
      .where('id', parseInt(id))
      .preload('company')
      .preload('model')
      .preload('area', function (builder) {
        builder.preload('city')
        builder.preload('zones')
      })
      .first()

    if (!device) {
      return Notify.error('Скутер не знайдено', {})
    }

    const deviceCommands = Config.get('commands')

    const area = await Area.query().where('id', device.areaId).preload('zones').preload('city').firstOrFail()
    const zoneColorsSettings = await Setting.findBy('name', 'zone_colors')
    const zoneColors = zoneColorsSettings?.value ? JSON.parse(zoneColorsSettings.value) : {}
    zonesList.forEach((zone) => {
      zone.color = zoneColors?.[zone.type] ?? zone.color
    })

    if (device.activeAt) {
      // if(moment().diff(moment(device.active_at), 'days') >= 1) {
      //   device.last_signal = moment(device.active_at).tz('Europe/Kiev').format('DD.MM.YYYY HH:mm')
      // } else {
      //   device.last_signal = moment.utc(moment().diff(moment(device.active_at), 'milliseconds')).format('H [hours],
m [minutes]') + ' ago'
      // }
    }

    if (device.batteryId) {
      await device.load('battery')
    }

    device.$extras.ridesTotalMileage = (await Database.from('rides').where('device_id',
device.id).sum('distance'))[0]?.sum ?? 0

    return view.render('Devices::devices/info', {
      pageTitle: i18n.formatMessage('admin.devices.devices'),
      breadcrumbs: [
        { name: '<i class="ki-duotone ki-home fs-3 text-gray-400"></i>', url: Route.makeUrl('admin.dashboard') },
        { name: i18n.formatMessage('admin.devices.list'), url: Route.makeUrl('Devices.devices.page') }
      ],
```

```
      device,
      area,
      deviceCommands,
      deviceStatuses: Device.getAllStatuses(),
      zonesList
    })
  }

  async edit({ view, params, auth, i18n }) {
    const { id } = params
    let device: any
    if (id) {
      device = await Device.query().where({ id }).preload('area').first()
      if (!device) {
        return Notify.error('Not found', {})
      }
    } else {
      device = new Device()
    }

    const { user: manager } = await auth.use('admin')

    const deviceModels = await DeviceModel
      .query()
      .select('device_models.*')
      .leftJoin('company_manager', 'device_models.company_id', 'company_manager.company_id')
      .where('company_manager.manager_id', manager.id)
      .where(function () {
        // if (!session.get('isGlobalSubaccountsMode')) {
        //   builder.where('company_manager.enabled', true)
        // }
      })
      .preload('area', function (builder) {
        builder.preload('company')
      })

    let deviceStatuses = Device.getAllStatuses()
    if (!id) {
      deviceStatuses = deviceStatuses.filter((s) => s.value === DeviceStatus.NOT_READY)
    } else {
      deviceStatuses = deviceStatuses.filter((s) => !this.ignoringStatuses.includes(s.value) || s.value === device.status)
    }

    return {
      modal: {
        title: id ? i18n.formatMessage('admin.devices.edit') : i18n.formatMessage('admin.devices.create_new'),
        content: await view.render('Devices::devices/form', {
          device,
          deviceModels,
          deviceStatuses
        }),
        cancel: i18n.formatMessage('admin.general.cancel'),
        submit: i18n.formatMessage('admin.general.save')
      },
      success: true
    }
  }

  async save({ request, auth }) {
    const isUpdateMode = typeof request.input('id') !== 'undefined'

    const saveDeviceSchema = schema.create({
```

```
  id: schema.number.optional(),
  device_model_id: schema.number(),
  uid: schema.string(),
  number: schema.string(),
  status: schema.string(),
  sim: schema.string.optional({}, [
    rules.maxLength(64)
  ]),
  iccid: schema.string.optional()
})

const {
  id,
  device_model_id,
  uid,
  number,
  status,
  sim,
  iccid
} = await request.validate({ schema: saveDeviceSchema })

let device: any // Device | null

if (isUpdateMode) {
  device = await Device.find(id)
  if (!device) {
    return Notify.error('Vehicle not found', {})
  }

} else {
  if (await Database.from('devices').where('uid', uid).whereNull('deleted_at').first()) {
    return Notify.error('Скутер з таким IOT вже зареєстрований')
  }

  const deviceDb = await Device.query().where('number', number).first()
  if (deviceDb && !deviceDb.deletedAt) {
    return Notify.error('Скутер з таким номером вже зареєстрований', {})
  }

  device = deviceDb || new Device()
  device.uid = uid
  device.locked = true
  device.status = DeviceStatus.NOT_READY
}

const deviceModel = await DeviceModel.find(device_model_id)
if (!deviceModel) {
  return Notify.error('Device model not found', {})
}

device.number = number
device.deviceModelId = deviceModel.id
device.areaId = deviceModel.areaId
device.companyId = deviceModel.companyId
if (sim) {
  device.sim = sim
}
if (iccid) {
  device.iccid = iccid
}
device.deletedAt = null
```

```
    const { user: manager } = await auth.use('admin')

    if (id && device.status !== status) {
      if (this.ignoringStatuses.includes(status) || (this.ignoringStatuses.includes(device.status))) {
        return Notify.error('Wrong status', {})
      } else {
        const currentStatus = device.status

        device.status = status

        await DeviceStatusChangeLog.create({
          adminId: manager.id,
          adminEmail: manager.email,
          deviceId: device.id,
          deviceNumber: device.number,
          statusFrom: currentStatus,
          statusTo: device.status,
          coordinates: {
            lat: device.lat,
            lng: device.lng
          }
        })

        const executeCommands = async (commands) => {
          for (let i = 0; i < commands.length; i++) {
            DeviceCommunicator.execute(device.uid, commands[i], {
              userId: manager.id,
              userEmail: manager.email
            }).then(() => { }).catch(() => { })

            if (i !== commands.length - 1) {
              await new Promise(resolve => {
                setTimeout(resolve, 2000)
              })
            }
          }
        }

        if ([DeviceStatus.TRANSPORTATION, DeviceStatus.MAINTENANCE,
DeviceStatus.READY].includes(status)) {
          executeCommands(['UNLOCK']).then(() => { }).catch((e) => console.log(e))
        }
        else if ([DeviceStatus.TRANSPORTATION, DeviceStatus.MAINTENANCE,
DeviceStatus.READY].includes(currentStatus)) {
          executeCommands(['LOCK', 'RESET']).then(() => { }).catch((e) => console.log(e))
        }
      }
    }

    // if(!isUpdateMode) {
    Event.emit('updateBrokerDevices', {})
    // }

    await device.save()

    return Notify.success('Saved', {})
  }

  async delete({ params }) {
    const { id } = params
    const device = await Device.find(Number(id))
```

```
  if (!device) {
    return Notify.error('Something went wrong. Vehicle not found', {})
  }

  // const { user: manager } = await auth.use('admin')

  // if (manager.id !== 1) {
  //   return Notify.error('Permission denied')
  // }

  await device.delete()

  return Notify.success('Deleted', {})
}

async sendCommand({ params, request, auth }) {
  const { id, cmd } = params

  const device = await Device.find(id)
  if (!device) {
    return Notify.error('Device not found')
  }

  const { user: manager } = await auth.use('admin')

  DeviceCommunicator.execute(device.uid, cmd, {
    userId: manager.id,
    userEmail: manager.email,
    ...request.all()
  }).then(() => { }).catch(() => { })

  return Notify.success('Command was sent')
}

async editStatus({ request, i18n, view }) {
  const input = request.all()
  // const rules = {
  //   device_ids: 'required|array'
  // }

  // const validation = await validate(input, rules)
  //
  // if (validation.fails()) {
  //   return response.json(Notify.error(validation.messages()[0].message, {}))
  // }

  const devices = await Device.query().whereIn('id', input.device_ids).whereNotIn('status', this.ignoringStatuses)
  if (!devices.length) {
    return Notify.error('All of the selected scooters are on the ride or have specific status')
  }

  const deviceStatuses = Device.getAllStatuses().filter((s) => !this.ignoringStatuses.includes(s.value))

  return {
    modal: {
      title: `Edit status`,
      content: await view.render('Devices::devices/status-edit/form', {
        devices,
        deviceStatuses,
        callback: input.callback || null
      }),
      cancel: i18n.formatMessage('admin.general.cancel'),
```

```
        submit: i18n.formatMessage('admin.general.save')
      },
      success: true
    }
  }

  async saveStatus({ request, acl, auth }) {
    const input = request.all()

    // const rules = {
    //   device_ids: 'required|array',
    //   'device_ids.*': 'number',
    //   status: 'required'
    // }
    //
    // const validation = await validate(input, rules)
    // if (validation.fails()) {
    //   return response.json(Notify.error(validation.messages()[0].message, {}))
    // }

    if (!Device.getAllStatuses().filter((s) => !this.ignoringStatuses.includes(s.value)).map((s) =>
s.value).includes(input.status)) {
      return Notify.error('Wrong status')
    }

    const query = Device.query().whereIn('id', input.device_ids)

    if (!acl.hasPermissions('devices.edit')) { // devices_ch_status TODO
      query.whereNotIn('status', this.ignoringStatuses)
    }

    const devices = await query

    const { user: manager } = await auth.use('admin')

    for (const device of devices) {
      const currentStatus = device.status

      device.status = input.status
      await device.save()

      await DeviceStatusChangeLog.create({
        adminId: manager.id,
        adminEmail: manager.email,
        deviceId: device.id,
        deviceNumber: device.number,
        statusFrom: currentStatus,
        statusTo: device.status,
        coordinates: {
          lat: device.lat,
          lng: device.lng
        }
      })

      const executeCommands = async (commands) => {
        for (let i = 0; i < commands.length; i++) {
          DeviceCommunicator.execute(device.uid, commands[i], {
            userId: manager.id,
            userEmail: manager.email
          }).then(() => { }).catch(() => { })

          if (i !== commands.length - 1) {
```

```
          await new Promise(resolve => {
            setTimeout(resolve, 2000)
          })
        }
      }
    }

    if ([DeviceStatus.TRANSPORTATION, DeviceStatus.MAINTENANCE,
DeviceStatus.READY].includes(input.status)) {
      executeCommands(['UNLOCK']).then(() => { }).catch((e) => console.log(e))
    }
    else if ([DeviceStatus.TRANSPORTATION, DeviceStatus.MAINTENANCE,
DeviceStatus.READY].includes(currentStatus)) {
      executeCommands(['LOCK', 'RESET']).then(() => { }).catch((e) => console.log(e))
    }
  }

  return Notify.success('Saved')
 }

 async editModel({ request, auth, i18n, view }) {
  const input = request.all()
  // const rules = {
  //   device_ids: 'required|array'
  // }
  //
  // const validation = await validate(input, rules)
  //
  // if (validation.fails()) {
  //   return Notify.error(validation.messages()[0].message, {})
  // }

  const ignoringStatuses = [DeviceStatus.IN_USE, DeviceStatus.RESERVED, DeviceStatus.PAUSE]
  const devices = await Device.query().whereIn('id', input.device_ids).whereNotIn('status',
ignoringStatuses).preload('model')

  if (!devices.length) {
   return Notify.error('All of the selected scooters are on the ride or have specific status')
  }

  const { user: manager } = await auth.use('admin')

  const deviceModels = await DeviceModel
   .query()
   .select('device_models.*')
   .whereIn('device_models.company_id', await manager.getEnabledCompanyIds())

  return {
   modal: {
    title: `Edit model`,
    content: await view.render('Devices::devices/model-edit/form', {
     devices,
     deviceModels
    }),
    cancel: i18n.formatMessage('admin.general.cancel'),
    submit: i18n.formatMessage('admin.general.save')
   },
   success: true
  }
 }

 async saveModel({ request }) {
```

```
    const input = request.all()

    // const rules = {
    //   device_ids: 'required|array',
    //   'device_ids.*': 'number',
    //   device_model_id: 'required|number'
    // }
    //
    // const validation = await validate(input, rules)
    // if (validation.fails()) {
    //   return Notify.error(validation.messages()[0].message, {})
    // }

    const deviceModel = await DeviceModel.find(input.device_model_id)
    if (!deviceModel) {
      return Notify.error('Device model not found')
    }

    const devices = await Device.query().whereIn('id', input.device_ids).whereNotIn('status', this.ignoringStatuses)

    for (const device of devices) {
      device.deviceModelId = deviceModel.id
      device.areaId = deviceModel.areaId
      device.companyId = deviceModel.companyId
      await device.save()
    }

    // Event.fire('updateBrokerDevices')

    return Notify.success('Saved')
  }

  async cancelReservation({ request }) {
    const { device_id } = request.all()

    const device = await Device.find(device_id)
    if (!device) {
      return Notify.error('Device not found')
    }

    if (device.status !== DeviceStatus.RESERVED) {
      return Notify.error('Device is not in reserve status')
    }

    const reservation = await Ride.query().where('device_id', device.id).where('status',
DeviceStatus.RESERVED).first()
    if (reservation) {
      reservation.status = RideStatus.RESERVATION_CANCELED
      await reservation.save()

      DeviceCommunicator.socket(reservation.userId, {
        action: 'ride_forced_finished',
        ride_id: reservation.id
      }).then(() => { }).catch((e) => console.log(e))

      if (!(await RideClass.hasActiveRidesByOrder(reservation.orderId))) {
        await OrderClass.complete(reservation.orderId)
      }
    }

    device.status = DeviceStatus.AVAILABLE
    device.reservedAt = null
```

```
    await device.save()

    return Notify.success('Success')
  }
}
```

DeviceForm.js

```
const ADM = require('../../../Adm/resources/assets/adm/adm')

ADM.modules.set('deviceForm', {

  init(context) {
    const modelSelect = $('select[name=device_model_id]', $(context)).select2()

    const setModelArea = () => {
      const selectedOption = $(modelSelect).find(':selected')

      if(selectedOption.data('area')) {
        const { name, company } = $(selectedOption).data('area')

        $('input[name="area_name"]', $(context)).val(name)
        $('input[name="company_name"]', $(context)).val(company.name)
      }
    }

    setModelArea()

    modelSelect.on('change', function() {
      setModelArea()
    })
  }
})
```

DeviceInfo.js

```
const ADM = require('../../../Adm/resources/assets/adm/adm')
const moment = require('moment')
const formatDate = (date) => moment(date).format('DD.MM.YY HH:mm:ss')

ADM.modules.set('deviceInfo', {

  events: {
    'click: #ridesSectionLink' : 'initDeviceRidesSection',
    'click: #activitySectionLink' : 'initDeviceActivitySection',
    'click: #notesSectionLink' : 'initDeviceNotesSection',
    'click: #changeDeviceStatus' : 'changeDeviceStatus',
    'click: #cancelReservation' : 'cancelReservation',
    'click: #commandsHistorySectionLink' : 'initCommandsHistorySection',
    'click: #problemReportsSectionLink' : 'initDeviceProblemReportsSection',
    'click: #batteryHistorySectionLink' : 'initDeviceBatteryHistorySection',
  },

  deviceRidesTableSelector: '#deviceRidesTable',
  deviceActivityTableSelector: '#deviceActivityTable',
  deviceNotesTableSelector: '#deviceNotesTable',
  deviceProblemReportsTableSelector: '#deviceProblemReportsTable',
  deviceBatteryHistoryTableSelector: '#deviceBatteryHistoryTable',

  map: {},
  zoneTypeColors: {},

  // Start movement history logic

  deviceHistoryTableSelector: '#deviceHistoryTable',
```

```
initMap(city) {
  const { lat: cityLat, lng: cityLng } = city

  const map = new google.maps.Map(document.getElementById('map'), {
    center: { lat: Number(cityLat), lng: Number(cityLng) },
    zoom: 16,
    disableDefaultUI: false,
    zoomControl: true
  })

  this.map = map

  return map
},

loadZones(area, map) {
  const { zones } = area

  for(const zone of zones) {
    new google.maps.Polygon({
      paths: zone.coordinates.map((i) => {
        return {
          lat: Number(i.lat),
          lng: Number(i.lng)
        }
      }),
      fillColor: this.zoneTypeColors[zone.type],
      fillOpacity: 0.35,
      strokeColor: this.zoneTypeColors[zone.type],
      strokeOpacity: 0.75,
      strokeWeight: 1.5,
      draggable: false,
      map: map,
      zoneId: zone.id,
      zoneType: zone.type
    })
  }
},

processMovementHistoryData(history) {
  const module = this
  const context = module.el()

  module.clearInstances()

  if(!history.length) {
    module.updateCurrentLocation()
    return true
  }

  const maxIndex = history.length - 1
  const slider = $('#movementHistorySlider', $(context))[0]
  noUiSlider.create(slider, {
    start: 0,
    step: 1,
    range: {
      min: 0,
      max: maxIndex
    },
    tooltips: [
      true
    ]
```

```
  })

  const coordinates = history.map((h) => {
    return {
      lat: Number(h.lat),
      lng: Number(h.lng)
    }
  })

  module.deviceHistoryTable = $(module.deviceHistoryTableSelector).DataTable({
    scrollY: '500px',
    scrollX: false,
    paging: false,
    ordering: false,
    data: history,
    columns: [
      {
        data: 'date',
        render: (data) => moment(data).format('DD MMM HH:mm:ss')
      },
      {
        data: 'charge',
        render: (data) => {
          if (data === undefined || data === null) {
            return ''
          }

          return `<span class="badge py-3 px-4 fs-7 badge-light-${data >= 50 ? 'success' : data >= 20 ? 'warning' :
'danger'}">${data}%</span>`
        }
      },
      {
        data: 'locked',
        render: (data) => {
          if(data === undefined || data === null) {
            return ''
          }

          return `<span class="badge py-3 px-4 fs-7 badge-light-${data ? 'danger' : 'success'}">${data ? 'OFF' :
'ON'}</span>`
        }
      },
      {
        data: 'noise',
        render: (data) => data ? 'Шум' : ''
      },
    ],

    drawCallback: (settings) => {
      const { nTable } = settings
      $(`tbody tr:eq(0)`, $(nTable)).css('background-color', '#F1F1F2')
    }
  })

  const deviceMarker = new google.maps.Marker({
    position: coordinates[coordinates.length - 1],
    title: formatDate(history[coordinates.length - 1].date),
    icon: {
      url: 'https://i.imgur.com/4EKn8e0.png'
    },
    map: module.map
  })
```

```
  const infoWindow = new google.maps.InfoWindow({
    content: `${coordinates[coordinates.length - 1].lat},${coordinates[coordinates.length - 1].lng}`
  })

  deviceMarker.addListener('click', function() {
    infoWindow.open(module.map, deviceMarker)
  })

  for (let i = 0; i < coordinates.length - 1; i++) {
    const polyline = new google.maps.Polyline({
      path: [coordinates[i], coordinates[i + 1]],
      strokeColor: '#0CB2FC',
      strokeOpacity: 1.0,
      strokeWeight: 2,
      zIndex: 1,
      map: module.map
    })

    const polylineBorder = new google.maps.Polyline({
      path: [coordinates[i], coordinates[i + 1]],
      strokeWeight: 5,
      strokeColor: '#186BCE',
      zIndex: 0,
      map: module.map
    })
    module.polylines.push(polyline)
    module.polylines.push(polylineBorder)
  }

  slider.noUiSlider.on('start.one', function (value) {
    value = Math.round(value)
    $('.noUi-tooltip', $(slider)).text(formatDate(history[value].date))
  })

  slider.noUiSlider.on('slide.one', function (value) {
    value = Math.round(value)
    $('.noUi-tooltip', $(slider)).text(formatDate(history[value].date))
    deviceMarker.setPosition(coordinates[value])
    deviceMarker.setTitle(formatDate(history[value].date))
    infoWindow.setContent(`${coordinates[value].lat},${coordinates[value].lng}`)
  })

  slider.noUiSlider.on('slide.end', function (value) {
    value = Math.round(value)

    const $scrollBody = $('.dataTables_scrollBody', $(context))
    $scrollBody.scrollTo( $(`${module.deviceHistoryTableSelector} tbody tr:eq(${value})`))

    $('tbody tr', $(module.deviceHistoryTableSelector)).css('background-color', 'transparent')
    $(`tbody tr:eq(${value})`, $(module.deviceHistoryTableSelector)).css('background-color', '#F1F1F2')
  })

  module.slider = slider
  module.deviceMarker = deviceMarker

  const { lat, lng } = coordinates[0] || {}
  if(lat && lng) {
    module.map.setCenter(new google.maps.LatLng(lat, lng))
  }
 },
```

```
getDeviceMovementHistory(from, to) {
  const that = this

  ADM.ajax({
    url: `/admin/devices/${that.device.id}/movement-history/list`,
    method: 'POST',
    dataType: 'json',
    data: {
      from,
      to,
      alertId: that.alertId
    },
    success: function (res) {
      let { history } = res
      history = history.filter((h) => h.lat && h.lng)

      // if (!history.length) {
      //   Swal.fire({
      //     text: 'History not found',
      //     icon: 'error',
      //     buttonsStyling: false,
      //     confirmButtonText: 'Close',
      //     customClass: {
      //       confirmButton: 'btn btn-primary'
      //     }
      //   })
      //
      //   return false
      // }

      that.processMovementHistoryData(history)
    }
  })
},

clearInstances() {
  if(this.slider) {
    this.slider.noUiSlider.destroy()
    this.slider = null
  }
  if(this.polylines.length) {
    for(const polyline of this.polylines) {
      polyline.setMap(null)
    }
  }
  if(this.deviceMarker) {
    this.deviceMarker.setMap(null)
  }

  if(this.deviceHistoryTable) {
    this.deviceHistoryTable.destroy()
    $('tbody', $(this.deviceHistoryTableSelector)).empty()
    this.deviceHistoryTable = null
  }

  this.lastLat = null
  this.lastLng = null
  this.lastUpdate = null
},

drawPolyline(start, end) {
  if(!start.lat || !start.lng || !start.date) {
```

```
      return false
    }

    const polyline = new google.maps.Polyline({
      path: [{ lat: start.lat, lng: start.lng }, { lat: end.lat, lng: end.lng }],
      strokeColor: '#0CB2FC',
      strokeOpacity: 1.0,
      strokeWeight: 2,
      zIndex: 1,
      map: this.map
    })

    const polylineBorder = new google.maps.Polyline({
      path: [{ lat: start.lat, lng: start.lng }, { lat: end.lat, lng: end.lng }],
      strokeWeight: 5,
      strokeColor: '#186BCE',
      zIndex: 0,
      map: this.map
    })

    this.polylines.push(polyline)
    this.polylines.push(polylineBorder)
  },

  updateCurrentLocation() {
    const that = this

    ADM.ajax({
      url: `/admin/devices/${that.device.id}/movement-history/current-geo`,
      method: 'GET',
      dataType: 'json',
      delay: 500,
      success: function(res) {
        let { success, lat, lng, date } = res
        if(!success) {
          return false
        }

        lat = Number(lat)
        lng = Number(lng)

        if(that.lastLat !== lat && that.lastLng !== lng) {
          that.drawPolyline({ lat: that.lastLat, lng: that.lastLng, date: that.lastUpdate }, { lat, lng, date })

          that.map.setCenter(new google.maps.LatLng(lat, lng))

          that.lastLat = lat
          that.lastLng = lng
          that.lastUpdate = date

          if(that.deviceMarker) {
            that.deviceMarker.setMap(null)
          }

          const deviceMarker = new google.maps.Marker({
            title: formatDate(date),
            position: {
              lat: Number(lat),
              lng: Number(lng),
            },
            icon: {
              url: 'https://i.imgur.com/4EKn8e0.png'
```

```
        },
        map: that.map
      })

      const infoWindow = new google.maps.InfoWindow({
        content: `${lat},${lng}`
      })

      deviceMarker.addListener('click', function() {
        infoWindow.open(that.map, deviceMarker)
      })

      that.deviceMarker = deviceMarker
    }
   },
   error: function() {}
  })
},

// End movement history logic

ReloadPage: function() {
  setTimeout(function() {
    location.reload()
  }, 500)
},

changeDeviceStatus(e, module) {
  ADM.ajax({
    url: '/admin/devices/edit-status',
    method: 'POST',
    dataType: 'json',
    data: {
      device_ids: [module.device.id],
      callback: 'deviceInfo::ReloadPage'
    }
  })
},

cancelReservation(e, module) {
  ADM.ajax({
    url: '/admin/devices/cancel-reservation',
    method: 'POST',
    dataType: 'json',
    data: {
      device_id: module.device.id
    },
    success: function(res) {
      const { type } = res.notification || {}
      if(type === 'success') {
        module.ReloadPage()
      }
    }
  })
},

generateQrCode: function (number) {
  const width = 120, height = 120

  new QRCode('qrcode', {
    text: `https://e-wings.com.ua/vehicles/${number}`,
    width: width,
```

```
    height: height,
    colorDark : '#000000',
    colorLight : '#ffffff',
    correctLevel : QRCode.CorrectLevel.H
   })

   const qrcode = $('#qrcode')
   const qrcodeCanvas = $('canvas', qrcode)[0]
   const dataURL = qrcodeCanvas.toDataURL()

   const resultCanvas = $('#canvas')[0]
   const ctx = resultCanvas.getContext('2d')

   const image = new Image()
   image.src = dataURL
   image.onload = function(){
    ctx.drawImage(this, 0, 0);
    qrcode.remove()
   }

   canvas.width = width
   canvas.height = height + 20
   ctx.fillStyle = 'white'
   ctx.fillRect(0, 0, canvas.width, canvas.height)

   ctx.fillStyle = 'black'
   ctx.font = '21px Verdana black'
   ctx.fillText(number , (canvas.width / 2) - (ctx.measureText(number).width / 2), height + 20)
  },

  initDeviceRidesSection: function (e, module) {
   if(module.deviceRidesTable) {
    return true
   }

   module.deviceRidesTable = $(module.deviceRidesTableSelector).DataTable({
    processing: true,
    serverSide: true,

    ajax: {
     headers: {
      'X-CSRF-TOKEN': $('meta[name="csrf-token"]').attr('content')
     },
     url: `/admin/devices/${module.device.id}/rides/list`,
     type: 'POST',
     dataType: 'json',
    },
    order: [0, 'desc'],
    autoWidth: false,
    scrollX: false,
    columns: [
     {
      data: 'id',
      name: 'rides.id',
      orderable: true,
      searchable: true,
      render(data) {
       return `<a class="nav-link fw-bold text-hover-primary" href="${ADM.url}/rides/show/${data}"
target="_blank">${data}</a>`
      }
     },
     {
```

```
  data: 'type',
  name: 'rides.type',
  orderable: true,
  searchable: true
 },
 {
  data: 'time_start',
  name: 'rides.time_start',
  orderable: true,
  searchable: true,
  render: (data) => data ? moment(data).format('DD.MM.YY HH:mm') : ''
 },
 {
  data: 'time_end',
  name: 'rides.time_end',
  orderable: true,
  searchable: false,
  render: (data) => data ? moment(data).format('DD.MM.YY HH:mm') : ''
 },
 {
  data: 'distance',
  name: 'rides.distance',
  orderable: true,
  searchable: true,
  render: (data, type, row) => {
   if(row.type === 'ride' && data) {
    return data
   }
   return ''
  }
 },
 {
  data: 'duration',
  orderable: false,
  searchable: false
 },
 {
  data: 'cost',
  name: 'rides.cost',
  orderable: true,
  searchable: true,
  render: (data, type, row) => {
   if(row.type === 'ride' && data) {
    return `${data} ₴`
   }
   return ''
  }
 },
 {
  data: 'stars',
  name: 'ride_feedback.stars',
  orderable: false,
  searchable: false
 },
 {
  data: 'comment',
  name: 'ride_feedback.comment',
  orderable: false,
  searchable: false
 },
 { data: 'user_name',
  name: 'users.name',
```

```
        render: (data, type, row) => `<a class="nav-link text-hover-primary no-history fw-bold"
href="${ADM.url}/users/show/${row.user_id}">${data}</a>`
       },
       {
        data: 'status',
        name: 'rides.status',
        orderable: true,
        searchable: true,
        render: (data) => {
         let className = ''
         switch (data) {
          case 'active':
           className = 'badge-light-success'
           break
          case 'finished':
           className = 'badge-light-danger'
           break
          default:
           className = 'badge-light-warning'
         }
         return `<span class="badge ${className}">${data}</span>`
        }
       }
      ]
     })
    },

    initDeviceNotesSection: function (e, module) {
     if(module.deviceNotesTable) {
      return true
     }

     module.deviceNotesTable = $(module.deviceNotesTableSelector).DataTable({
      processing: true,
      serverSide: true,
      pageLength: 10,
      autoWidth: false,

      ajax: {
       headers: {
        'X-CSRF-TOKEN': $('meta[name="csrf-token"]').attr('content')
       },
       url: `/admin/devices/${module.device.id}/notes/list`,
       type: 'POST',
       dataType: 'json',
      },
      order: [0, 'desc'],
      columns: [
       { data: 'id', name: 'device_notes.id', width: '5%' },
       { data: 'text', name: 'device_notes.text', width: '50%' },
       { data: 'admin_name', name: 'users.name', width: '20%' },
       { data: 'roles_names', name: 'roles.name', width: '20%' },
       {
        data: 'created_at',
        name: 'device_notes.created_at',
        width: '20%',
        render: (data) => moment(data).format('DD MMM yyyy HH:mm')
       }
      ]
     })
    },
```

```
initDeviceActivitySection: function(e, module) {
  if(module.deviceActivityTable) {
    return true
  }

  const context = module.el()

  const dateRangePicker = $('#dateRangePicker', $(context)).daterangepicker({
    buttonClasses: ' btn',
    applyClass: 'btn-primary',
    cancelClass: 'btn-secondary',
    locale: {
      format: 'DD/MMM/YYYY',
      firstDay: 1
    },
    startDate: moment(),
    endDate: moment().add(1, 'd'),
  })

  module.deviceActivityTable = $(module.deviceActivityTableSelector).DataTable({
    processing: true,
    serverSide: true,
    pageLength: 10,
    autoWidth: false,
    paging: false,
    orderable: false,
    searchable: false,
    bInfo: false,

    ajax: {
      headers: {
        'X-CSRF-TOKEN': $('meta[name="csrf-token"]').attr('content')
      },
      url: `/admin/devices/activity/list`,
      type: 'POST',
      dataType: 'json',
      data: function (data) {
        data.id = module.device.id

        data.from = dateRangePicker.data('daterangepicker').startDate.startOf('day').valueOf()
        data.to = dateRangePicker.data('daterangepicker').endDate.endOf('day').valueOf()
      }
    },
    columns: [
      {
        data: 'date',
        width: '20%',
        render: (data) => moment(data).format('DD.MM.yyyy HH:mm:ss')
      },
      { data: 'customer', name: 'users.name', width: '25%' },
      { data: 'action', name: 'users.name', width: '25%' },
    ]
  })

  $('button#refreshDeviceActivityTable').on('click', function() {
    module.deviceActivityTable.ajax.reload(null, false)
  })
},

initCommandsHistorySection: function(e, module) {
  const context = module.el()
```

```
  const dateRangePicker = $('#commandsHistoryDateRangePicker', $(context)).daterangepicker({
    timePicker: true,
    buttonClasses: ' btn',
    applyClass: 'btn-primary',
    cancelClass: 'btn-secondary',
    locale: {
      format: 'DD/MMM/YYYY HH:mm',
      firstDay: 1
    },
    startDate: moment().subtract(1, 'h'),
    endDate: moment(),
  })

  const $form = $('#commandsHistoryForm', $(context))
  $form.on('submit', function (e) {
    e.preventDefault()

    const uid = $('input[name="uid"]', $form).val()

    const from = dateRangePicker.data('daterangepicker').startDate.valueOf()
    const to = dateRangePicker.data('daterangepicker').endDate.valueOf()

    ADM.ajax({
      method: 'POST',
      url: '/admin/devices/commands-history',
      dataType: 'json',
      data: {
        uid,
        from,
        to
      },
      success: function (res) {
        if(res.success) {
          initTable(res.list)
          $('#commandsHistoryTableSection').removeClass('d-none')
        }
      }
    })
  })

  const initTable = (list) => {
    if (module.commandsHistoryTable) {
      module.commandsHistoryTable.destroy()
    }

    module.commandsHistoryTable = $('#commandsHistoryTable').DataTable({
      data: list,
      // responsive: true,
      sortable: true,
      paging: false,
      pageLength: 500,
      order: [[3, 'desc']],
      columns: [
        {
          data: 'type',
          render: (data, type, row) => {
            const capitalizeFirstLetter = (string) => string.charAt(0).toUpperCase() + string.slice(1)
            return `<span class="badge badge-light-${row.type === 'sent' ? 'success' :
'primary'}">${capitalizeFirstLetter(row.type)}</span>`
          }
        },
        {
```

```
          data: 'data',
          className: 'overflow-ellipsis mw-400px',
          render: (data, type, row) => {
            if (row.type === 'sent') {
              return row.command ? `${row.command}: ${JSON.stringify(row.data)}` : row.cmd
            }
            return row.cmd // Object.values(row.data || {}).length ? JSON.stringify(row.data) : row.cmd
          }
        },
        { data: 'provider' },
        {
          data: 'date',
          render: (data, type, row) => moment.utc(row.date).local().format('DD/MM/YYYY HH:mm:ss')
        }
      ]
    })
  }
 },

 initDeviceProblemReportsSection: function (e, module) {
  if(module.deviceProblemReportsTable) {
    return true
  }

  module.deviceProblemReportsTable = $(module.deviceProblemReportsTableSelector).DataTable({
    processing: true,
    serverSide: true,
    pageLength: 10,
    autoWidth: false,

    ajax: {
      headers: {
        'X-CSRF-TOKEN': $('meta[name="csrf-token"]').attr('content')
      },
      url: `/admin/devices/${module.device.id}/problem-reports/list`,
      type: 'POST',
      dataType: 'json',
    },
    order: [0, 'desc'],
    columns: [
      { data: 'id', name: 'device_problem_reports.id', width: '5%', searchable: false },
      {
        data: 'user_name',
        name: 'users.name',
        width: '10%',
        render: (data, type, row) => `<a class="nav-link text-hover-primary fw-bold"
href="${ADM.url}/users/show/${row.user_id}">${data}</a>`
      },
      {
        data: 'problem_types',
        name: 'device_problem_reports.problem_types',
        width: '20%',
        searchable: false,
        orderable: false,
        render: (data, type, row) => Array.isArray(row.problem_types) ? row.problem_types.map((pt) => `<span
class="badge badge-light me-2">${localisation[`admin.device_problem_reports.types.${pt}`]}</span>`).join('') : ''
      },
      { data: 'comment', name: 'device_problem_reports.comment', width: '30%' },
      {
        data: 'created_at',
        name: 'device_problem_reports.created_at',
        width: '20%',
```

```
        searchable: false,
        render: (data) => moment(data).format('DD MMM yyyy HH:mm')
       }
      ]
     })
    },

    initDeviceBatteryHistorySection: function(e, module) {
     if(module.deviceBatteryHistoryTable) {
      return true
     }

     module.deviceBatteryHistoryTable = $(module.deviceBatteryHistoryTableSelector).DataTable({
      processing: true,
      serverSide: true,

      order: [[0, 'desc']],

      ajax: {
       headers: {
        'X-CSRF-TOKEN': $('meta[name="csrf-token"]').attr('content')
       },
       url: `/admin/devices/${module.device.id}/battery-history/list`,
       type: 'POST',
       dataType: 'json'
      },
      columns: [
       { data: 'id', name: 'device_battery_histories.id', orderable: false, searchable: false },
       {
        data: 'battery_number',
        name: 'batteries.number',
        orderable: true,
        searchable: true,
        render: (data, type, row) => `<a class="nav-link fw-bold text-hover-primary"
href="${ADM.url}/batteries/show/${row.battery_id}">${data}</a>`
       },
       {
        data: 'installation_time',
        name: 'device_battery_histories.installation_time',
        orderable: false,
        searchable: false,
        render: (data) => {
         if(!data) {
          return ''
         }
         // <span class="text-gray-800 fw-bold mb-1 fs-6">${row.installer_employee_name ||
row.default_employee_name || ''}</span>

          return `<div class="d-flex justify-content-start flex-column">
                <span class="text-gray-800 fw-semibold d-block fs-7">${moment(data).format('DD.MM.yyyy
HH:mm:ss')}</span>
               </div>`
        }
       },
       {
        data: 'removal_time',
        name: 'device_battery_histories.removal_time',
        orderable: false,
        searchable: false,
        render: (data) => {
         if(!data) {
          return ''
```

```
          }
          // <span class="text-gray-800 fw-bold mb-1 fs-6">${row.remover_employee_name || "}</span>

          return `<div class="d-flex justify-content-start flex-column">
                    <span class="text-gray-800 fw-semibold d-block fs-7">${moment(data).format('DD.MM.yyyy
HH:mm:ss')}</span>
                  </div>`
        }
       }
     ]
   })
 },

 init(context) {
  const module = this

  this.deviceRidesTable = null
  this.deviceActivityTable = null
  this.deviceNotesTable = null
  this.deviceProblemReportsTable = null
  this.deviceBatteryHistoryTable = null

  this.polylines = []
  this.lastLat = null
  this.lastLng = null
  this.lastUpdate = null

  this.device = JSON.parse($('input#device', $(context)).val())
  const zonesList = JSON.parse($('input#zonesList', $(context)).val())
  this.zoneTypeColors = Object.fromEntries(zonesList.map((z) => [z.type, z.color]))
  this.area = JSON.parse($('input#area', $(context)).val())
  this.city = this.area.city

  if(this.device.lat && this.device.lng) {
   const map = this.initMap(this.city)
   this.loadZones(this.area, map)

   const startInterval = () => {
    module.realTimeInterval = setInterval(function() {
     module.updateCurrentLocation()
    }, 15 * 1000)
   }

   module.updateCurrentLocation()

   function cb(start, end) {
    if(end.diff(start, 'minutes') <= 5) {
     $('.date-range-title', $periodSelect).html('Realtime')

     module.clearInstances()
     module.updateCurrentLocation()
     startInterval()
    }
    else {
     $('.date-range-title', $periodSelect).html(start.format('MMM D HH:mm') + ' - ' + end.format('MMM D
HH:mm'))
     clearInterval(module.realTimeInterval)
     module.getDeviceMovementHistory(moment.utc(start).format(), moment.utc(end).format())
    }
   }

   const start = moment().subtract(5, 'minutes')
```

```
    const end = moment()

    // Select period
    const $periodSelect = $('#periodSelect', $(context)).daterangepicker({
      timePicker: true,
      startDate: start,
      endDate: end,
      ranges: {
        'Realtime': [moment().subtract(5, 'minutes'), moment()],
        'Last 10 minutes': [moment().subtract(10, 'minutes'), moment()],
        'Last 30 minutes': [moment().subtract(30, 'minutes'), moment()],
        'Last 1 hour': [moment().subtract(1, 'hour'), moment()],
        'Last 2 hours': [moment().subtract(2, 'hour'), moment()],
        'Last 6 hours': [moment().subtract(6, 'hour'), moment()],
        'Last 12 hours': [moment().subtract(12, 'hour'), moment()],
        'Last 24 hours': [moment().subtract(24, 'hour'), moment()],
        'Last 48 hours': [moment().subtract(48, 'hour'), moment()],
        'Last 72 hours': [moment().subtract(72, 'hour'), moment()]
      }
    }, cb)

    cb(start, end)
  }

  if(this.device.number) {
    this.generateQrCode(this.device.number)
  }
 }
})
DeviceModelForm.js
const ADM = require('../../../Adm/resources/assets/adm/adm')

ADM.modules.set('deviceModelForm', {

 initDynamicPrice() {
  const module = this
  const context = module.el()

  const $dynamicPricesToggleInput = $('input[name="tariffs[has_dynamic_prices]"]')

  const deviceModelId = $('input[name="id"]', $(context)).val()
  const hasDynamicPrices = $dynamicPricesToggleInput.is(':checked')

  const toggleDynamicPrices = () => {
    const $dynamicPricesContainer = $('#dynamicPricesContainer')

    $dynamicPricesContainer.toggleClass('d-none', !$dynamicPricesToggleInput.is(':checked'))
  }

  toggleDynamicPrices()
  $dynamicPricesToggleInput.on('change', function() {
    toggleDynamicPrices()
  })

  const $dynamicPricesSettingsRepeater = $('#dynamicPricesSettingsRepeater', $(context)).repeater({
    initEmpty: true,

    defaultValues: {
      weekday: 1,
      price: 0,
      price_unlock: 0,
      price_pause: 0
```

```
},

show: function () {
  $(this).slideDown()

  $('input.schedule-item', $(this)).daterangepicker({
    timePicker: true,
    timePicker24Hour: true,
    timePickerIncrement: 15,
    locale: {
      format: 'HH:mm'
    },
  }).on('show.daterangepicker', function (ev, picker) {
    picker.container.find('.ranges').hide()
    picker.container.find('.calendar-table').hide()
    picker.container.find('.drp-selected').hide()
  })
},

hide: function (deleteElement) {
  $(this).slideUp(deleteElement)
```

ВІДГУК

керівника на бакалаврський кваліфікаційний проект
“ Система моніторингу електросамокатів ”
ШВЕЦЬ Іван Андрійович

Бакалаврський кваліфікаційний проект виконано у поєднанні із дисципліною „Алгоритмізація та програмування” , в межах якої розглядаються питання побудови розгалужених алгоритмів та побудови програм по цим алгоритмам.

У кваліфікаційному проєкті студента Швець І.А. Зроблений аналітичний огляд літературних джерел по основам проєктування системи моніторингу електросамокатів. Найбільш популярним є метод створення бездротової інтернет мережі моніторингу електросамокатів. Розглянута будова і управління моніторингу електросамокатів. Кваліфікаційний проєкт присвячений дослідженню IoT-рішень для мобільного транспорту, розробці клієнт-серверного вебдодатку на основі стеку технологій (AdonisJS, WebSocket, Tailwind, Google Maps API), створенню бази даних, побудові інтерфейсу адміністратора, реалізації телеметрії в реальному часі, а також візуалізації пристроїв на карті з фільтрами та індикаторами стану. В цілому під час роботи над кваліфікаційним проєктом студент Швець І.А. проявив себе як грамотний спеціаліст в галузі телекомунікацій, показав вміння та навички і набуті компетентності в дослідженні методів побудови телекомунікаційної мережи за технологією розумного будинку.

Кваліфікаційний проєкт виконано на високому технічному рівні, він має безперечну актуальність в області сучасних телекомунікацій, а студент Швець І.А. заслуговує оцінки «відмінно».

Доцент кафедри телекомунікацій, медійних та інтелектуальних технологій **Мішан В.В.**

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНИЙ ПРОЕКТ

Дипломник: *Швець Іван Андрійович*
Тема роботи: *Система моніторингу електросамокатів*
Спеціальність *172 «Телекомунікації та радіотехніка»*

Обсяг кваліфікаційного проект

Кількість листів креслень 5 Кількість сторінок записки 84

1. Короткий зміст роботи та прийнятих рішень в результаті виконаного наукового дослідження *Кваліфікаційний проєкт присвячений розробці інтелектуальної системи моніторингу електросамокатів, що базується на технологіях Інтернету речей (IoT). У межах роботи проведено аналіз сучасних рішень у сфері мікромобільності та систем трекінгу, обґрунтовано вибір архітектури багаторівневої клієнт-серверної системи з використанням GPS-модулів, контролерів ESP8266 та цифрового зв'язку через протокол MQTT. Розроблено структуру системи, реалізовано програмне забезпечення для збору, передачі та візуалізації телеметричних даних у вебінтерфейсі. Проведено моделювання змінних сценаріїв руху, аналіз роботи вебсервісу з інтеграцією Google Maps API та протестовано систему в умовах реального часу. Отримані результати підтверджують ефективність запропонованого рішення для моніторингу стану та місцезнаходження електросамокатів у міському середовищі.*

2. Висновок про відповідність роботи дипломному завданню *Каліфікаційний проект відповідає виданому завданню*

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки та техніки і передових методів роботи: *Кваліфікаційний проєкт виконаний на високому науково-технічному рівні з використанням сучасних досягнень у галузі Інтернету речей, мікроконтролерних систем, цифрового керування та комп'ютерного моделювання. У вступній частині обґрунтовано актуальність теми з урахуванням зростаючої популярності мікромобільності та необхідності ефективного моніторингу електросамокатів у міських умовах. У теоретичному розділі проаналізовано сучасні IoT-рішення для транспортної сфери, наведено порівняння апаратних і програмних платформ для реалізації систем трекінгу. У проєктній частині запропоновано архітектуру системи моніторингу електросамокатів із використанням мікроконтролера ESP8266, GPS-модуля, протоколу MQTT та вебінтерфейсу для візуалізації даних. У розділі моделювання наведено результати комп'ютерної симуляції та тестування передачі координат у*

реальному часі, з урахуванням варіативності руху та змін у навантаженні на сервер. Усі етапи роботи виконано послідовно, з дотриманням сучасних технічних вимог та стандартів у сфері розробки IoT-систем.

4. Позитивні сторони роботи: *Робота присвячена актуальному напряму розвитку інтелектуальних транспортних рішень — розробці ефективної IoT-системи моніторингу електросамокатів у міському середовищі. Автор демонструє глибоке розуміння принципів побудови мікроконтролерних систем, засвоєння сучасних підходів до збору, передачі та візуалізації телеметричних даних. Особливої уваги заслуговує інтеграція цифрового керування на базі ESP8266, використання протоколу MQTT та Google Maps API, що відповідає сучасним тенденціям у сфері Інтернету речей. Робота має високу практичну цінність і може бути використана як основа для подальших розробок у сфері розумного міського транспорту та мікромобільності.*

5. Негативні сторони роботи: *Робота в цілому виконана на достатньо високому рівні, однак має деякі недоліки. Зокрема, експериментальна частина є обмеженою — результати тестування системи представлені переважно у вигляді базової телеметрії, без детального аналізу роботи пристрою в умовах реального міського середовища (наприклад, у зоні нестабільного покриття або при змінній швидкості руху). Також у роботі не проведено порівняння із комерційними або альтернативними системами моніторингу мікромобільності, що могло б посилити обґрунтованість обраної архітектури.*

6. Оцінка графічного оформлення та пояснювальної записки роботи: *немає*

7. Відгук про роботу в цілому: *У цілому кваліфікаційний проєкт виконаний якісно, має чітку структуру, логічну послідовність викладення матеріалу та демонструє належний рівень технічної деталізації. Робота засвідчує вміння здобувача застосовувати сучасні інженерні підходи в галузі Інтернету речей, мікроконтролерних систем та вебтехнологій. Отримані результати є теоретично обґрунтованими і мають практичну цінність для впровадження у сфері моніторингу мікромобільного транспорту. Робота відповідає вимогам, що ставляться до кваліфікаційних проєктів освітнього рівня бакалавра. бакалавра.*

8. Інші зауваження: *немає*

9. Оцінка дипломної роботи: *Кваліфікаційний проєкт відповідає встановленим вимогам і заслуговує оцінки відмінно (5.00/A), а її автору Швецю Івану, присвоєння кваліфікації бакалавра зі спеціальності «Телекомунікації та радіотехніка»*

10. Рецензент (прізвище, ім'я, по батькові, місце роботи) _____

Нестун Віктор Миколайович
канд. техн. наук, доц., доцент кафедри кібербезпеки

«13» червня 2025р.

підпис

Завідувачу кафедри телекомунікацій,
медійних та інтелектуальних технологій
д.т.н., професору ПІДЧЕНКУ Сергію
здобувача вищої освіти
Швеця Івана
ФІТ, гр. ТР2с-22-1

ЗАЯВА

З правилами чинного Положення про систему забезпечення академічної доброчесності у Хмельницькому національному університеті, згідно з яким виявлення академічного плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів академічної відповідальності, ознайомлений (а). Про використання спеціалізованих програмних засобів (СПЗ) StrikePlagiarism та Anti-Plagiarism для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність академічного плагіату оповіщений (а). Надаю університету право на передачу мого кваліфікаційного проєкту для обробки та збереження в базах даних СПЗ і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються СПЗ.

Також надаю свою згоду на обробку й збереження університетом мого кваліфікаційного проєкта «Система моніторингу електросамокатів» в Інституційному репозитарії Хмельницького національного університету.

Робота надається для перевірки в електронному варіанті. Електронна версія мого кваліфікаційного проєкту збігається (ідентична) з друкованою.

29 травня 2025 р.
дата

підпис

Fri Jun 06 08:58:57 EEST 2025, Степюк Віктор Іванович, Хмельницький національний університет, ХНУ

Anti-Plagiarism (UA) v-15.281 Educational

The maximum coincidence with one document 8.0%

Dictionaries check: en_US, ru_RU, ua_UA. **Errors in the documents: 14%**

ID: 243811 Title: Система моніторингу електросамокатів Added in a DB: 2025-06-06 Authors: Швець Іван Андрійович Heads: Міщан Віктор Володимирович Consultants: Opponents:	Document		Sum coincidence on the DB	
	Symbols	Lexemes	Symbols	Lexemes
	55174	850	7943 (14%)	127 (15%)

Plagiarism sources

ID	Description	Plagiarism presence in the document	
		Symbols	Lexemes

Протокол аналізу звіту подібності експертом

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Іван ШВЕЦЬ

Співавтор:

Назва: Система моніторингу електросамокатів

Експерт: Віктор МІШАН, к.т.н., доц.

Підрозділ: Кафедра телекомунікацій, медійних та інтелектуальних технологій

Коефіцієнт подібності 1:33.5%

Коефіцієнт подібності 2:28.2%

Мікропробіли: 0

Заміна букв: 0

Інтервали: 0

Білі знаки: 0

Дата створення звіту: 2025-06-06 10:27:48.0

Після аналізу Звіту подібності констатую наступне:

☐ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

Дата 6.06.25 експерт Пивовар О.С

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ КАФЕДРИ ТМІТ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Назва кваліфікаційної роботи: Система моніторингу електросамокатів
Автор Швець Іван Андрійович
Освітня програма: «Телекомунікацій, медійні технології та інтелектуальні мережі»
Рівень вищої освіти Бакалавр
Спеціальність 175«Телекомунікацій та радіотехніка»
Науковий керівник: к.т.н., доцент Мішан В.В

На основі аналізу кваліфікаційної роботи на дотримання вимог академічної доброчесності (у т.ч. відсутності ознак академічного плагіату) з урахуванням результатів перевірки роботи спеціалізованим програмним засобом(ами) комісія зробила такий висновок:

№	Висновок	Позначка про відповідність
1	Ознаки академічного плагіату	
1.1	Запозичення, виявлені в роботі, є законними і не є академічним плагіатом (далі – зазначаються підстави віднесення запозичень до правомірних, якщо потрібно). Робота приймається до захисту.	
1.2	Виявлені запозичення не є академічним плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи (далі – зазначаються детальні та аргументовані підстави віднесення запозичень до правомірних). Робота приймається до захисту, але має бути відкоригована.	+
1.3	Виявлені запозичення не є академічним плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота може бути допущена до захисту після того як буде відкоригована та доопрацьована і успішно пройде повторну перевірку на академічний плагіат.	
1.4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття текстових запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
2	Інші види порушень академічної доброчесності	

Підтвердження:
Виявлені запозичення взяті з різних джерел і не перевищують допустимий обсяг

Дата 9.06.2025 р.

Завідувач кафедри Підпис Пірченко С.К Ім'я, ПРІЗВИЩЕ

Гарант освітньої програми Підпис Сотенко В.І. Ім'я, ПРІЗВИЩЕ

Керівник кваліфікаційної роботи Підпис Мішан В.В. Ім'я, ПРІЗВИЩЕ

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ГОЛОВІ ЕКЗАМЕНАЦІЙНОЇ КОМІСІЇ

Направляється студент Швець Іван Андрійович (прізвище, ім'я, по батькові) на захист дипломного проєкту (роботи)

за спеціальністю 172 - Телекомунікації та радіотехніка

На тему: Система моніторингу електросамокатів

Дипломний проєкт (робота), рецензія і довідка про перевірку на плагіат додаються.

Декан факультету (підпис) Наталія ГОВОРУЩЕНКО (ім'я, прізвище)

ДОВІДКА УСПІШНОСТІ

Швець І. А. за період навчання на факультеті інформаційних технологій з 2022 по 2025 роки повністю виконав навчальний план спеціальності з таким розподілом оцінок за:
національною шкалою: відмінно 41,67 %, добре 41,67 %, задовільно 16,67 %.
шкалою ЄКТС: A 37,21 %, B 25,58 %, C 18,60 %, D 11,63 %, E 6,98 %.

Методист факультету (підпис) Тетяна Федорук (ім'я, прізвище)

ВИСНОВОК КЕРІВНИКА ДИПЛОМНОГО ПРОЄКТУ (РОБОТИ) ТА ОБГРУНТУВАННЯ ОЦІНКИ

Студент Швець Іван Андрійович виконував кваліфікаційний проект на тему „Система моніторингу електросамокатів". В ході виконання проекту розглянуті особливості роботи телекомунікаційної мережі в системі роботи електросамокатів. Розроблено алгоритм роботи системи моніторингу та програмне забезпечення яке виконане на високому технічному рівні. За результатами виконаної проекту здобувачу оцінку „відмінно"

Оцінка дипломного проєкту (роботи)

Керівник дипломного проєкту (підпис) Віктор В. Міщан (ім'я, прізвище)

"09" червня 2025 р.

ВИСНОВОК КАФЕДРИ ПРО ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ)

Дипломний проєкт (роботу) розглянуто. Студент Швець І. А. допускається до захисту цього проєкту (роботи) в екзаменаційній комісії.

Завідувач кафедри ТКМ (назва)

Федченко С.К. (підпис, ім'я, прізвище)

"09" 06 2025 р.